72 October 1982 $2.50 ($3.00 in Canada)

Dr. Dobb's Journal

For Users of Small Computer Systems

Portable Pidgin for the Z80

68000 Cross Assembler

8086 Graphics Subroutines

and even more!

includes the DDJ CP/M Exchange!

Dr. Dobb's Journal
For Users of Small Computer Systems

October 1982 Volume 7, Issue 10

Subscription Rates: $25 per year within the United States; $44 for first class to Canada and Mexico; $62 for airmail to other countries. Payment must be in U.S. Dollars, drawn on a U.S. Bank.

Writer's Guidelines: All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh, dark ribbon* on continuous white paper. Please avoid printing on perforations. Payment is in contributor's copies. Requests to review galleys must accompany the manuscript when it is first submitted. Authors may receive a copy of the complete writer's guidelines by sending a self-addressed, stamped envelope.

Donating Subscribers **Contributing Subscriber:** $50/year ($25 tax deductible). **Retaining Subscriber:** $75/year ($50 tax deductible). **Sustaining Subscriber:** $100/year ($75 tax deductible). **Lifetime Subscriber:** $1000 ($800 tax deductible). **Corporate Subscriber:** $500/year ($400 tax deductible, receives *five* one-year subscriptions).

Contributing Subscribers: Thomas H. Grohne, G. V. Elkins, Robert M. Connors, William J. McEown, Robert C. Luckey, Robert N. Stabler, DeWitt S. Brown, John B. Palmer, Burks A. Smith, Transdata Corporation, Mark Ketter, John W. Campbell, Friden Mailing Equipment, Howard E. Decker, Jim Wilson, Frank Lawyer, Chris Pettus, Paul Lutus, Ben Goldfarb, Rodney Black, Steven Weiss, Pat Phelan, John Brodie, James Fleming, Unison World, Thomas Davis, Jan Steinman, Ronald E. Johnson, G. Hunter, Kenneth Drexler, Real Paquin, Ed Malin, John Saylor, Jr. **Sustaining Subscribers:** Pete Roberts, Steven Fisher Enterprizes. **Lifetime Subscriber:** Michael S. Zick.

Foreign Distributors **UK & Europe:** Hofacker-Verlag, Tegernseer Strass 18, D-8150 Holzkirchen, West Germany. **Asia & Australia:** ASCII Publishing Co., Ltd, Aoyama Building 5F, 5-16-1 Minami Aoyama, Minato-ku, Tokyo 107, Japan.

CONTENTS

Articles

Departments

Dr. Dobb's Journal (USPS 307690) is published twelve times per year by People's Computer Company, Box E, 1263 El Camino Real, Menlo Park, CA 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025 · 415/323-3111

QUNIX™

AT LAST—Multi-User and Multi-Tasking for the IBM PC

QUNIX is the total business, scientific, educational, and graphic Operating System for the IBM Personal Computer. Now you can unleash the full power of the IBM Personal Computer. The total capability of your computer is now available for future growth.

QUNIX—For the first time, the tremendous power of the QUNIX single-user single-tasking operating system is now available with MULTI-USER and MULTI-TASKING for up to 8 simultaneous users and up to 250 simultaneous tasks.

QUNIX—Now allows high speed arithmetic, high speed and high resolution graphics by supporting the 8087 coprocessor for number crunching.

QUNIX—The ideal business operating system designed for transaction and word processing that can be shared by up to 8 simultaneous users. No additional computer required . . . just add keyboards and terminals to the basic system.

QUNIX—Gives you mainframe/mini computer Data Processing capabilities. Features such as Hierarchical Files, Password Protection, 16-Character File and Directory Names, Data Communications to all mainframe, minicomputer and microcomputers, File Security and Winchester Support are all built into the fabulous QUNIX operating system.

QUNIX—After the frustration and limitations of DOS and CP/M™, you will LOVE QUNIX.

QUANTUM SOFTWARE SYSTEMS, INC.
7219 Shea Court
San Jose, California 95139
Tel: (408) 629-9402

QUANTUM SOFTWARE SYSTEMS, LTD.
P.O. Box 5318, Station "F"
Ottawa, Ontario, Canada K2C 3H5
Tel: (613) 235-4297

PC/SOFTWARE, INC.
926 Natoma Street,
San Francisco, CA 94103
(415) 863-0652

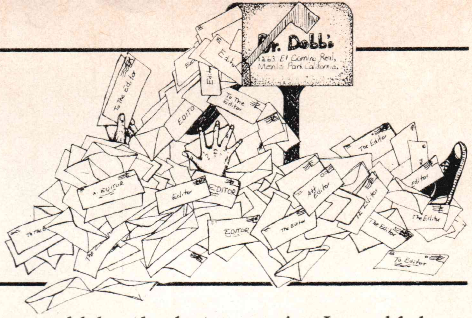
I am a ☐ User ☐ Dealer ☐ Distributor
☐ Please send me a brochure

Name ______________ Title ______________
Company ______________
Address ______________
City ______________ State ______ Zip ______
Computer ______________ Disk Format ______________

IBM-Trade Mark of IBM Corporation
CP/M-Trade Mark of Digital Research, Inc
QUNIX-Trade Mark of Quantum Software Sys...

Circle no. 195 on reader service card.

LETTERS



Dear Gary Kildall . . .

Dear *Dr. Dobb's*,

I found the article comparing CP/M-86 with MS-DOS (and CP/M-80) quite interesting, though I certainly agree with the conclusion that it is disappointing that Digital Research and Microsoft can't come up with anything better than a copy of CP/M-80. In my opinion, the biggest item missing from all three of these systems is I/O redirection.

Recently, Digital Research ran ads in a newspaper in this area for people to work on CP/M for the Motorola 68000. Are we in for yet another copy of CP/M-80? That would be rather like running a Ferrari engine on one cylinder. (It is gratifying, however, that Digital Research has finally recognized that someone besides Intel makes microprocessors!) I would like to suggest to Digital Research that they use Microware's OS-9^{tm} operating system (for the 6809) as a model for CP/M-68K (or whatever they call it), rather than giving us yet another copy of CP/M-80.

Sincerely,
Jim Howell
5472 Playa Del Rey
San Jose, CA 95123

Micro Compiler in Brobdingnag

Dear Marlin Ouverson,

Enclosed is an article and program submitted for your review and inclusion in *Dr. Dobb's*, if you so desire. Let me add that I enjoy your magazine very much, have all the issues from #1, and it would be the last magazine I would drop. (I get over thirty engineering or computing magazines.)

I have modified the Small-C compiler to work on a 370 under VM. How's that for reverse engineering? It also has FOR, GOTO, two-dimension arrays, etc.

Regards,
Chris L. Torkildson
13791 Heywood Ct.
Apple Valley, MN 55124

Dear Mr. Ouverson,

I have been using Small-C since the Software Works first offered it for CP/M. I recently got enough memory to allow it to compile itself correctly, and have started implementing all the changes suggested

(Continued on page 59)

Editorial

Going Public . . .

Dr. Dobb's Journal has always championed the cause of public-domain software. It was accidentally founded by a group of idealistic hackers who only wanted to write and publish a tiny Basic interpreter. Their challenge was to fit it in about 4K; their commitment was to put it in the public domain.

This idea – doing good work and letting the public have it with no strings – was so good that readers who managed to get their hands on a copy just wouldn't let *DDJ* finish its intended three-issue life cycle. Instead, seven years later it is the landmark publication for systems software.

One reason for practically giving away perfectly good programs is to enable individuals to build a library of tools for their own use. In the larger sense, of course, commercial products would be greatly enhanced by drawing on some of the truly superb items that have been launched into the public domain. In fact, a number of good packages for sale these days may find their roots in a *Dr. Dobb's Journal* listing, or some other non-proprietary source.

This altruistic influence on end-user technology has, we think, improved the state of the art. The *DDJ* community waves the public-domain flag because we think it is important. With this kind of material, the available computing power is greater than would otherwise be true.

Many generous and sympathetic authors should be thanked for making *DDJ*'s mission successful. In addition to our present contributors, we remember the work of Dennis Allison, Steve Wozniak, Gary Kildall, Tom Pittman, Ron Cain, Li-Chen Wang and Ward Christensen. They are among the many fine people who have used *DDJ* to make important contributions to the microcomputing community.

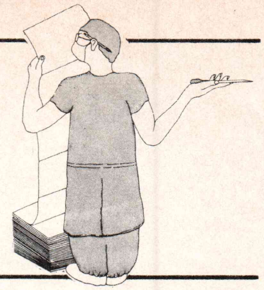
Unfortunately, the creation of public-domain software is not as popular a pastime as it once was. Maybe, in the beginning, we had more of a sense of community – wonderful machines with unlimited potential, but with nothing more helpful than a row of switches on the front panel. And if you think today's documentation is bad . . . it took a major team effort just to get the things operating.

Now, software protection schemes seem to get more creative thought than the programs they were designed to protect. Instead of sharing resources, one sees even (or perhaps especially) inexperienced programmers becoming canny to any possible profit to be gained or lost. Fortunately, there is still a body of people willing to share their work for just the good feeling, or the prestige, or the common cause of more public-domain software.

In the interest of building the public domain, we encourage authors not to reserve rights to programs published in *DDJ*, even though we will publish ones which reserve commercial rights. If the technical growth potential of computers is going to be realized, it won't be by those large companies with heavy investments in current technology – it will be by individuals working together to push beyond the limits of today's product line. When you hear one of us refer to the *DDJ community*, that's what we mean.

– Marlin Ouverson

DR. DOBB'S CLINIC



by Dave Cortesi

The Very Last SUBMIT Item

For months we have been droning on about the SUBMIT command of CP/M, telling you how useful it can be and showing how to increase its usefulness with fixes and small utilities. In June we showed the utilities PAUSE and BEEP, which increase the operator's control over a submitted job. In August we brought you Robert Pasky's nicely-integrated set of patches that make SUBMIT handle lowercase input, control characters and null lines correctly. With them applied, you can submit command streams to ED and PIP for unattended execution. In September we presented Don Wright's QUITIF program, which gives the designer of a SUBMIT file the ability to check for some errors.

Now, courtesy of Digital Research, we SUBMIT users have the ultimate in SUBMIT patches. The July/August issue of *Microsystems* reproduces, unedited and without commentary, a set of applications notes, the work of the Technical Support Group at Digital Research, Inc. One of these is a lengthy patch which causes SUBMIT to append its output to an existing $$$.SUB file, instead of replacing that file. The result is that we can include a SUBMIT command in a submitted job. The inner submit file will be appended to the active submit file; when it completes, the outer submit file will be resumed with the line following the SUBMIT command.

Nested submits! And they come so easily, too. When SUBMIT runs, it places the submitted commands, one per 128-byte record, in reverse order in the file $$$.SUB. This was originally done, we think, for the convenience of the Console Command Processor (CCP). The CCP consumes records from $$$.SUB from the end of the file toward the beginning. That lets it use the "record count" byte of the directory entry for $$$.SUB as its pointer to the next record of the file. It couldn't keep such a pointer in storage, because there is no location in storage that is proof against overlay by an application program.

But this use of $$$.SUB makes the file work like a push-down stack. The CCP pops records off the end of the file. The new patch makes SUBMIT push records onto the end of the file, rather than erasing the file and building a whole new one. Nesting of submitted jobs follows automatically.

The Digital Research patch, with our own comments, is shown in Listing 1 (page 58). Assemble it, then use the I and R commands of DDT to overlay its hex file onto a copy of SUBMIT.COM (one with the Pasky Patches applied to it). The updated SUBMIT command does just about everything we could ask of it, so with this item we are turning the whole matter of CP/M SUBMIT over to Gene Head and the *CP/M Exchange*.

Debugging the Debuggers

We've had some letters from people who have problems with the Digital Research debug utilities, DDT and ZSID. L. Barker sent us a note on two holes in ZSID. It doesn't interpret the LDIR instruction correctly when the operands (the two strings defined by the HL, DE and BC registers) overlap on each other. The LDIR (long move) instruction of the Z80 copies a source string into a target string. When the target string overlaps on the source string (as when the source begins at 0200h, the target at 0201h, and the length is 255 bytes), LDIR becomes a Replicate instruction, duplicating the leading byte of the source into the target over and over. ZSID doesn't handle it right, though; it stops after copying one byte.

Barker also noted that the disassembler in ZSID can't seem to handle the "LD A,R" and "LD R,A" instructions. Its List command won't display them and its Assemble command won't accept them.

Gavin Brickell of Auckland, New Zealand, thought that he'd found a bug in DDT, but he hadn't. It appeared that DDT was modifying a word in storage without any reason. However, it had a reason: the word in question was the top word of the stack as defined by the program under test. DDT uses one word on the program's stack in at least three points during the execution of the Go command. The remarkable thing is that DDT manages to get by with only one word of the program's stack; it seems not to need any more.

Dana Trout of Goleta, CA, wrote in with a method for changing the number of the RST vector that DDT uses for breakpoints. One of the Digital Research applications notes mentioned above covers the same area. We think that DDT's use of RST 7 should be left alone if at all possible. DDT (and SID, and ZSID, and any other debugging tool) is going to use *some* restart vector, and RST 7 is the one that is documented as being for this use. The great majority of CP/M systems will leave RST 7 free for debuggers. A modified debugger, exported to another system, is quite likely to crash that system by conflicting with a legitimate interrupt vector. Furthermore, there's a good reason for using RST 7 for debug breakpoints: most hardware will return FFh when asked to read from RAM that doesn't exist. FFh is the opcode for an RST 7 instruction; if that is the breakpoint vector, a wild branch to nonexistent memory will cause an automatic breakpoint!

Finally, Nick Hammond of Pebble Beach, CA, contributed a cute trick that can be played with DDT. DDT could be useful for poking around in the innards of CP/M, if only it would leave things alone. "The problem with DDT," Hammond writes, "is that it changes things when it gets control. Location 5 no longer points to the BDOS except via a circuitous series of jumps, and the CCP is missing, having

been overwritten." Hammond supplies a technique for getting a copy of DDT into the transient area below the CCP, and getting control into it with the BDOS vector and the CCP intact. It works like this.

First, invoke DDT as a command, with its own .COM file as argument:

```
A>ddt ddt.com
```

DDT will start up and load DDT.COM. Give the Go command with no address; control will enter the loaded copy. It will relocate itself just below the first-started copy of DDT, clear of the CCP, put its own vector into the RST 7 location, and prompt for a command. Use the Assemble or Substitute command to put an RST 7 instruction at location 0100h. Then use the G0 or a control-c to do a warm start. The CCP will be reloaded over the first copy of DDT, but the second copy will remain. Then create and call a null command file as a means of causing a branch to location 100h, where you stored a RST 7:

```
A>save 0 null.com
A>null
```

Control will pass to the RST 7 at location 100, which will return control to the second copy of DDT. You can now use it to peer about the system, observing low storage and the CCP in their native states.

Ask Uncle

"I have a Diablo 1620," says Ernest Knipp of Houston, TX, "and the Courier-10 printwheel has a pound-sterling symbol. I can't get this symbol to print. Is there any way to reach this unreachable character?"

You betcha, Ernest. There are 96 spokes on the Diablo printwheel. Ninety-four of them print in response to the 94 printing chracters of the ASCII code. Two of the spokes correspond to the ASCII space (32h) and the DEL (7Fh) codes. Those two can't be reached by sending the printer a single ASCII code. When it gets a space, the printer just advances the carriage; it doesn't use the printwheel at all. And it conforms to the ASCII standard by ignoring DEL characters entirely.

However, the manual for our 1650 printer says that the sequence ESCape, "Y" will print the spoke matching 32h, and ESCape, "Z" will print the one matching 7Fh. In other words, if your printer receives the *two* bytes 1Bh, 59h, it should print the sterling symbol. If it receives the bytes 1Bh, 5Ah, it should print a logical-not symbol.

Assorted Grumps

A person who would rather remain anonymous has told us that he had it from an equally anonymous, but supposedly well-informed, source that the code of the IBM PC's BASIC interpreter was indeed produced by an 8080-to-8086 translator program. That's all very

(Continued on page 58)

Circle **no. 120** on reader service card.

68000 Cross Assembler

The June 1982 issue of *DDJ* contained a brief description of the computer system which is being developed by myself, Darryl Uchitil, and Jim Hannas as a test bed for system software and hardware using the Motorola 68000 microprocessor. The reader response to both *DDJ* and myself as a result of that article has been interesting and informative to Jim, Darryl and myself, and I would like to thank all of you who have called or written me since June. I will present at the end of this article a summary of answers to some of the most frequently asked questions about our system for the benefit of those who have not tried to contact me or who have written but have not yet received my reply.

Cross-Assembler Description

This article will describe the cross assembler that was mentioned in the June 1982 issue of *DDJ*. It was developed because we needed a way to write small programs for execution on a single 68000 processor to test the hardware and to test the interface between the 68000 processors and the I/O bus control processor. The cross assembler for the 68000 has been run on two different PDP-11 systems, one running RT-11, and one running RSX-11M. It has also been run on a VAX/VMS system under the applications migration executive. It should be possible to adapt the assembler to other machines by recoding assembly-language subroutines for 32-bit arithmetic and modifying the routines that reference octal constants, but as of this writing, no attempt has been made to do so.

by Al Kossow

Allen Kossow, Medical College of Wisconsin, Department of Physiology, 8701 Watertown Plank Road, Milwaukee, WI 53226.

The cross assembler consists of a main program and a collection of subroutines, each performing part of the assembly process. The cross assembler is a two-pass assembler; the source code is read twice by the program. The first pass through the source code picks up all of the labels, and the second pass generates the listing and object files. All operations to files are handled by individual subroutines. These subroutines are SOURCE, LIST, and OBJECT. All machine-dependent routines for opening and closing of files are located in these routines.

Each of the three routines for the source, listing and object files is called when the program is first started, to determine where each is to be sent. Some checking is done to determine if just a carriage-return was typed, so that there is a default of only a listing to the terminal and no object file is established. Once the source for the assembler and the destinations of the listing and object are established, the assembler starts the assembly by setting the pass number to one and calls the subroutine PARSE.

PARSE scans the input line and determines if the statement is just a comment. If it is not a comment, it splits apart the label, opcode and operand fields of the statement, and returns pointers to these fields to be used by the statement evaluation routines.

After the line is parsed, the statement opcode is evaluated by the subroutine PRCESS. PRCESS calls DECOPC, which takes the characters pointed to by the opcode pointer from PARSE and attempts to find an opcode that matches. If an opcode is found, the values of one or more skeletal opcodes (opcodes without effective addresses or sizes) are returned.

After evaluating the opcode, the operands are evaluated to determine their general type. All "simple" operands, such as registers, are evaluated immediately, while "complex" operands (i.e., operands containing labels) are not evaluated.

If a valid opcode skeleton was returned, there will be a number returned by DECOPC which represents a general way in which to evaluate this type of operation. This number is used by PRCESS in a multi-way branch to different sections of code for evaluation of the opcode. Once in the specific section of code for the opcode, the operands are checked for validity in the opcode, and if the operand was "complex" it is evaluated and a value is returned.

Each specific section of code for an opcode builds up the opcode skeleton, filling in the size and effective address fields as necessary for that type of instruction. The result of this evaluation is an array of 16-bit values which represent the result of that line of the assembly, and a count of the number of words generated by that line.

At the end of the opcode processing routine is the section of code that handles labels. If a label was detected and is valid for the opcode type, the current value of the location counter is placed in the symbol table entry for that label.

If this is the second pass, the object and listing is generated based on the values in the instruction word array; otherwise, the next line is fetched and the evaluation process is repeated. After the last line of the program is read, the input file is closed, and a routine is called to print the contents of the symbol table.

Frequently Asked Questions About The Multi-68000 System

Q: Can I get a copy of your assembler?

A: If you don't feel like typing in the source code in this article, I can send you the source as it appears here on an 8" single-sided, single-density floppy disk in either CP/M or RT-11 format for $25 on an as-is, no-support basis. I can provide the source on a CP/M disk, but I have no way to compile it or test it there.

Q: What is CLICS? Where can I get it?

A: CLICS is a collection of subroutines for 2-D graphics developed by Mike Garrett while he was with the Department of Defense. Since the time I wrote the article, I have heard that Mike left DOD and started a company selling graphics software written in C. I have not been able to reach him to determine if this is true or not. The CLICS software I have been working with came off of the 1980 Spring DECUS RSX-11 special interest group symposium tape. This tape is available from the DECUS library.

Q: When/where can I buy one of your systems?

A: The notion of selling this system has been discussed a number of times between the three of us, but we just don't want to get into the computer business. The schematics and blank PC boards will probably be available from us on an as-is, no-support basis, along with the software we develop, but we have no plans to sell or support systems.

Q: Why didn't you build your system on a "XYZ" bus?

A: I had expected much more flack about using the PUNIBUS than I actually received. The primary reason for not using a commercial bus was the desire to build a system which had a few large (8" by 15") cards to minimize the number of cards to build. We also probably would not have used a special bus if we had planned on selling the system once we finished it.

In closing, I would like to thank all of you that have taken the time to call or write me about our project, and will promise to keep you informed of our progress in future issues of *DDJ*.

DDJ

(Listing begins on page 14)

Reader Ballot
Vote for your favorite feature/article in each issue.
Circle Reader Service **No. 520**

68000 Cross Assembler (XASM)

(Text begins on page 12)

Due to its length, this listing will be continued next month.

Listing One

```
      PROGRAM MAIN
C
C     M68000 CROSS-ASSEMBLER MAIN PROGRAM
C
C
C DATE:
C     FEBRUARY 10, 1981
C
C REVISION:
C     X1.0    (EXPERIMENTAL PRE-RELEASE)
C

C
C AUTHOR:
C             Allen Kossow
C             Medical College of Wisconsin
C             Department of Physiology
C             8701 Watertown Plank Rd.
C             Milwaukee, WI 53226
C
C PGM NOTES:
C
C             Symbols are unique to 8 chrs.
C             Symbol table size is currently set to 512.
C             Symbol table size is determined by the memory constraints
C             of the machine it's running on. The current symbol table
C             size works under RT-11 with 64kb of memory and under RSX-11.
C
C             This program follows most of the conventions of the
C             Motorola 68000 assembler with the following exceptions:
C
C             1) no macros or pc relative addressing (except branches)
C             2) generalized (i.e., simple) object file format
C             3) listing format similar to that of MACRO-11
C             4) doesn't handle adr regs as special cases (i.e., MOVEA is just a MOVE).
C
C


C.... LOGICAL UNIT DEFINITION
C     1 = SOURCE FILE
C     2 = OBJECT FILE
C     3 = LIST   FILE
C     5 = KEYBD
C
C

      IMPLICIT INTEGER (A-Z)
      BYTE NAME(8),SYMFLG(513)
```

(Continued on column 2)

```
C
      COMMON /LST/ LUNIT,PASS,NAME,NOPAGE,NOLINE,MEFLG,IERCNT
C
      COMMON /SYMT/STIND,SYMADR,PC,NOSYM,NEWPC,SYMFLG
C
      COMMON /PRSE/ OPPTR,MODPTR,OPNPTR,LABEL,CMTPTR
     +,PRFLG,SCANPT,OPCLEN,OPNPT2,IMODE
C
      COMMON /SRC/ LNELEN,ISERR,NOCARD,SRCLNE
C
      COMMON /OBJOUT/ OBJBUF,OBJWC,LFLG,RFLG,DBFLG
C
      COMMON /OPCPTS/ OPTYP,OPSKEL,OPSK2,OPIDX
C
      COMMON /CNVT   / WORD,PL
C
      COMMON /HEXFLG/ ENDFLG,HEXWC,HEXPC,OLDPC
C
      DIMENSION OBJBUF(40)
C
      INTEGER*4 PC,NEWPC,SYMADR(512),HEXPC,OLDPC
C
      BYTE ERR,SRCLNE(81),LABEL(8),PL(132)
C
C.... TELL FORTRAN TO IGNORE INTEGER OVERFLOWS ON
C     MULTIPLY AND DIVIDE.
C

C
      ERR=128
C
C.... THE FOLLOWING CALL IS NO OP'ED OUT FOR F4P
C
C     CALL SETERR(1,ERR)
C
C.... INITIALIZE VARIABLES
C
5     NOPAGE=0
      RFLG=1
      LFLG=1
C
C.... OPEN FILES
C
      CALL SOURCE(1)
      CALL LIST(1)
      CALL OBJECT(1)
C
C.... DO PASS 1
C
1     NOSYM=0
      PASS=1
      CALL I4CLR(PC)
      DO 10 I=1,8
      NAME(I)=32
10    CONTINUE
```

(Continued on column 3)

```
C
C.... READ ONE LINE OF SOURCE FILE
C
15    CALL I4CLR(NEWPC)
      CALL SOURCE(2)
C
C.... IF EOF DETECTED DO PASS 2
C
      IF(ISERR.EQ.1) GOTO 20


C
C.... RESET MULTIPLE ERROR FLG
C
      MEFLG = 0
C
C.... PARSE SOURCE LINE
C
      CALL PARSE
C
C.... IF NULL LINE GET NEXT LINE
C
      IF(PRFLG.EQ.0) GOTO 15
C
C.... PROCESS SOURCE LINE
C
      CALL PRCESS
C


C.... IF END DETECTED DO PASS 2 ELSE GET NEXT LINE
C
      IF(ISERR.EQ.1) GOTO 20
      I=JADD(PC,NEWPC,PC)
      GOTO 15
C
C.... DO PASS 2
C
C
C.... REW SOURCE SET TO PASS 2 AND RESET PC
C
20    CALL SOURCE(3)
      PASS=2
      IERCNT = 0
      CALL I4CLR(PC)
      CALL I4CLR(HEXPC)
      CALL I4CLR(OLDPC)
C
C.... FLUSH PRINT BUFFER IN CASE ANYTHING LEFT
C.... FROM LAST ASSEMBLY
C
      DO 25 I=1,132
25    PL(I) = '40
C
C.... INITIALIZE OBJECT BUFFER
C
```

(Continued on page 16, column 1)

68000 Cross Assembler (XASM)

(Listing continued, text begins on page 12)

Listing One

```
      ENDFLG = 0
      HEXWC  = 0
C
C....  PRINT FIRST PAGE HEADING
C
      CALL NEWPAG
30    CALL I4CLR(NEWPC)
      OBJWC = 0
      CALL SOURCE(2)
C
C....  EOF DETECTED
C
      IF(ISERR.EQ.1) GOTO 50
C
C....  RESET MULTIPLE ERROR FLG
C
      MEFLG = 0
C
C....  PARSE LINE
C
      CALL PARSE
C
C....  PRINT A LINE OF ONLY COMMENTS NORMALLY
C
      IF(CMTPTR.EQ.1) GOTO 40
C
C....  CHECK FOR PARSING ERRORS
C
      IF(PRFLG.EQ.0) GOTO 30
C
C....  PROCESS IT
C
38    CALL PRCESS
C
C....  GENERATE LISTING
C
40    CALL LSTLNE
C
C....  CHECK IF THERE IS OBJ CODE TO GENERATE
C
      IF(OBJWC.EQ.0) GOTO 45
      CALL BLDOBJ
C
C....  DO NEXT LINE IF NOT END
C
45    IF(ISERR.EQ.1) GOTO 50
      I=JADD(PC,NEWPC,PC)
      GOTO 30
C
C....  END OF ASSEMBLY, OUTPUT BALANCE OF OBJ BUFFER
C
200   ISERR=0
      READ(1,210,END=250) (SRCLNE(I),I=1,80)
210   FORMAT(80A1)
      NOCARD=NOCARD+1
C
C CONVERT ALL CHARACTERS
C
      DO 225 I=1,80
      IF(SRCLNE(I).GE.32) GO TO 220
215   SRCLNE(I)=32
      GO TO 225
220   IF(SRCLNE(I).LT.96) GO TO 225
      SRCLNE(I)=SRCLNE(I)-32
      IF(SRCLNE(I).GE.96) GO TO 215
225   CONTINUE
C
C REMOVE TRAILING BLANKS
C
      LNELEN=80
230   IF(SRCLNE(LNELEN).NE.32) GO TO 240
      LNELEN=LNELEN-1
      IF(LNELEN.GT.0) GO TO 230
240   LNELEN=LNELEN+1
      SRCLNE(LNELEN)=0
      GO TO 500
C

C END OF FILE
C
250   ISERR=1
      GO TO 500
C
C REWIND SOURCE FILE
C
300   REWIND 1
      NOCARD=0
      GO TO 500
C
C CLOSE SOURCE FILE
C
400   CLOSE(UNIT=1)
500   RETURN
      END
      SUBROUTINE LIST(LCODE)
C
C PERFORMS OPEN AND CLOSE ON LIST FILE
C
C INPUT:LCODE = 1 => OPEN FILE (NAME READ FROM KEYBOARD)
C               2 => CLOSE FILE
C
      BYTE FILNAM(12)
      INTEGER PASS
      BYTE NAME(8)
C
      COMMON /LST/ LUNIT,PASS,NAME,NOPAGE,NOLINE,MEFLG,IERCNT
C
      COMMON /FNAM/ FILNAM,OBJFLG
C INPUT:
C     ICODE = 1 => FIND OPERAND IN SYMBOL TABLE.  IF NOT FOUND,
C                  IT IS ENTERED INTO THE TABLE AS REFERENCED
C                  BUT NOT DEFINED.  THE INDEX OF THE SYMBOL
C                  IN THE SYMBOL IS RETURNED IN 'STIND'.
C
C             2 => FIND LABEL IN SYMBOL TABLE.  IF FOUND AND ALREADY
C                  DEFINED AND THIS IS THE FIRST PASS OF THE
C                  ASSEMBLER, THE MULTIPLE DEFINED BIT IS SET IN
C                  SYMFLG.  IF FOUND BUT ONLY PREVIOUSLY REFERENCED,
C                  THE DEFINED BUT PREVIOUSLY REFERENCED BIT IS SET
C                  AND THE REFERENCED BIT IS CLEARED.  IF NOT FOUND,
C                  IT IS ENTERED AND THE DEFINED BIT IS SET.
C
C     IADDR =      ADDRESS OF SYMBOL FOR ENTERING INTO SYMBOL TABLE.
C     SYMBOL=      SYMBOL TO LOOK UP OR ENTER IN SYMBOL TABLE.
C
C OUTPUT:
C       STIND = INDEX INTO SYMBOL TABLE FOR SYMBOL.
C
C FORMAT OF 'SYMFLG':

C
C BIT   MEANING IF SET
C  0    SYMBOL HAS BEEN REFERENCED BUT NOT DEFINED.
C  1    SYMBOL HAS BEEN DEFINED AND WAS REFERENCED BEFORE DEFINITION.
C  2    SYMBOL HAS BEEN DEFINED AND THERE WERE NO REFERENCES BEFORE.
C  3    SYMBOL HAS BEEN MULTIPLE DEFINED.
C  4    SYMBOL IS AN EQUATED VALUE
C
      IMPLICIT INTEGER (A-Z)
      BYTE SYMFLG(513),SYMSTR(8),SRCLNE(81)
      DIMENSION SYMSYM(4,512),SYMBOL(4),SYMLIN(512)
      INTEGER*4 SYMADR(512),IADDR
      INTEGER*4 PC,NEWPC
      COMMON/SYMT/STIND,SYMADR,PC,NOSYM,NEWPC,SYMFLG
      BYTE NAME(8)
      COMMON /LST/ LUNIT,PASS,NAME,NOPAGE,NOLINE,MEFLG,IERCNT
      COMMON /SRC/ LNELEN,ISERR,NOCARD,SRCLNE
      COMMON/SYMN/SYMSYM,SYMLIN
C
C     PACK SYMBOL TWO BYTES TO A WORD
C
      DO 100 J=1,4
      I = J*2
100   SYMBOL(J) = ((SYMSTR(I-1)*256).OR.SYMSTR(I))
C
C     SEARCH FOR SYMBOL IN SYMBOL TABLE
C
      STIND  = 1
      MOVFLG = 0
      IF(NOSYM.EQ.0) GO TO 200
      DO 120 STIND=1,NOSYM
      DO 110 J=1,4
      IF(SYMSYM(J,STIND).NE.SYMBOL(J)) GO TO 115
110   CONTINUE
      GO TO 300
115   DO 118 J=1,4
```

```
C
50      ENDFLG = 1
        CALL BLDOBJ
C
C....   PRINT SYMBOL TABLE
C
        CALL PST
C
C....   CLOSE FILES AND DO IT AGAIN
C
        CALL SOURCE(4)
        CALL LIST(2)
        CALL OBJECT(2)
        GOTO 5
        END
        SUBROUTINE SOURCE(ICODE)
C
C PERFORMS ALL OPERATIONS OF SOURCE INPUT FILE
C
C INPUT:
C ICODE = 1 => OPEN SOURCE FILE (NAME READ FROM KEYBOARD)
C         2 => READ ONE LINE FROM SOURCE FILE INTO
C              'SRCLNE' (80R1 FORMAT). TRAILING BLANKS
C               ARE DELETED. ZERO CHAR IS INSERTED AT
C               THE END OF THE LINE.
C         3 => REWIND SOURCE FILE.
C         4 => CLOSE SOURCE FILE.
C
C OUTPUT:
C       SRCLNE = SOURCE LINE FOR CODE 2
C       LNELEN = LENGTH OF LINE FOR CODE 2
C       ISERR  = 1 IF END OF FILE ON READ (ZERO OTHERWISE)
C       NOCARD = CARD NUMBER READ FROM SOURCE (1-?)
C
        BYTE FILNAM(12)
        BYTE SRCLNE(81)
        COMMON/SRC/LNELEN,ISERR,NOCARD,SRCLNE
        COMMON /FNAM/ FILNAM,OBJFLG
C
C SELECT FUNCTION
C
        GO TO (100,200,300,400),ICODE
C
C OPEN SOURCE FILE
C
100     TYPE 110
110     FORMAT('$Src file name: ')
        READ (5,120) ICNT,FILNAM
120     FORMAT(Q,12A1)
        IF(ICNT.EQ.0) STOP
        CALL ASSIGN(1,FILNAM,ICNT)
        NOCARD=0
        GOTO 500
C
C READ SOURCE LINE
```

(Continued on page 16, column 2)

```
C
C SELECT FUNCTION
C
        GO TO (100,200),LCODE
C
C....   ASSIGN DEFAULT LISTING TO CONSOLE
C
100     LUNIT=5
        TYPE 110
110     FORMAT('$Lst file name: ')
        READ (5,115) ICNT,FILNAM
115     FORMAT(Q,12A1)
        IF(ICNT.EQ.0) GOTO 116
C
C....   IF THERE IS A FILENAME ASSIGN LISTING TO LUN 3
C
        LUNIT = 3
        CALL ASSIGN(LUNIT,FILNAM,ICNT)
116     NOPAGE=0
        GO TO 300
C
C CLOSE FILE
C
200     IF(LUNIT.EQ.5) RETURN
        CALL CLOSE(LUNIT)
300     RETURN
        END
        SUBROUTINE OBJECT(ICODE)
C
C PERFORMS OPEN AND CLOSE ON OBJECT FILE
C
        BYTE FILNAM(12)
        INTEGER PASS
        BYTE NAME(8)
        COMMON /LST/ LUNIT,PASS,NAME,NOPAGE,NOLINE,NEFLG,IERCNT
        COMMON /FNAM/ FILNAM,OBJFLG
        GOTO (100,200),ICODE
100     TYPE 110
110     FORMAT ('$Obj file name: ')
        READ (5,115) ICNT,FILNAM
115     FORMAT(Q,12A1)
        IF(ICNT .EQ.0) GOTO 116
        CALL ASSIGN(2,FILNAM,ICNT)
        OBJFLG = 1
        RETURN
116     OBJFLG = 0
        RETURN
200     IF(OBJFLG.EQ.0) RETURN
        CALL CLOSE(2)
        RETURN
        END
        SUBROUTINE SYMTBL(ICODE,IADDR,SYMSTR)
C
C SYMBOL TABLE PROCESSOR
C
```

(Continued on page 16, column 3)

```
        IF (SYMSYM(J,STIND).LT.SYMBOL(J)) GOTO 120
        IF (SYMSYM(J,STIND).EQ.SYMBOL(J)) GOTO 118
        MOVFLG = 1
        GOTO 200
118     CONTINUE
120     CONTINUE
C
C SYMBOL WAS NOT FOUND
C
200     IF(NOSYM.LT.513) GO TO 210
        CALL ERROR(221)
        STIND=513
        GOTO 400
210     IF (MOVFLG.EQ.0) GOTO 218
        ITEMP = NOSYM
211     DO 212 J=1,4
212     SYMSYM(J,ITEMP+1) = SYMSYM(J,ITEMP)
        CALL JMOV (SYMADR(ITEMP),SYMADR(ITEMP+1))
        SYMFLG(ITEMP+1) = SYMFLG(ITEMP)
        SYMLIN(ITEMP+1) = SYMLIN(ITEMP)
        ITEMP = ITEMP - 1
        IF (ITEMP.GE.STIND) GOTO 211
218     NOSYM = NOSYM + 1
        DO 220 J = 1,4
220     SYMSYM (J,STIND) = SYMBOL(J)
        IF(ICODE.EQ.1) GO TO 250
        SYMFLG(STIND)=4
        CALL I4CLR(SYMADR(STIND))
        I=JADD(SYMADR(STIND),IADDR,SYMADR(STIND))
        SYMLIN(STIND) = NOCARD
        GOTO 400
250     CALL I4CLR(SYMADR(STIND))
        SYMFLG(STIND)=1
        SYMLIN(STIND) = 0
        GOTO 400
C
C SYMBOL FOUND
C
300     IF(PASS.EQ.2.OR.ICODE.EQ.1) GOTO 400
        IF(SYMFLG(STIND).NE.1) GO TO 310
        SYMFLG(STIND)=2
        CALL I4CLR(SYMADR(STIND))
        I=JADD(SYMADR(STIND),IADDR,SYMADR(STIND))
        SYMLIN(STIND) = NOCARD
        GOTO 400
310     SYMFLG(STIND)=SYMFLG(STIND).OR.8
400     RETURN
        END
        SUBROUTINE CNVHEX(INDEX)
C
C CONVERTS 4 BITS TO HEX ASCII AND INSERTS INTO 'PL' AT 'INDEX'
C
C INPUT: WORD = VALUE
C        INDEX= WHERE TO INSERT IN PL
C
```

(Continued on next page)

68000 Cross Assembler (XASM)

(Listing continued, text begins on page 12)

Listing One

```
C OUTPUT:
C       WORD = WORD/16
C
        BYTE PL(132),DIG
        INTEGER WORD
        COMMON /CNVT/ WORD,PL
        CALL GETBIT(WORD,DIG)
        PL(INDEX)=DIG
        RETURN
        END
        SUBROUTINE INSDAT(IPL,IDIG)
C
C CONVERTS BINARY DATA TO HEX ASCII AND INSERTS INTO 'PL'
C
C INPUT:IPL = INDEX TO INSERT INTO PL
C       IDIG= NUMBER OF DIGITS TO CONVERT AND INSERT
C       WORD= VALUE TO CONVERT (IN COMMON - NOT REFERENCED HERE)
C
        I=IDIG
5       J=IPL+I-1
        CALL CNVHEX(J)
        I=I-1
        IF(I.LE.0) RETURN
        GO TO 5
        END

        SUBROUTINE IHX(ISZ,IDTA,IPPOS)
C
C PRINT A 4 OR 8 DIGIT HEX VALUE
C NUMBER OBTAINED STARTING AT 'WORD'
C AND PUT INTO PRINT BUFFER 'PL' STARTING IN COL 1
C
        IMPLICIT INTEGER (A-Z)
        COMMON /CNVT/ WORD,PL
        COMMON /LST/LUNIT,PASS,NAME,NOPAGE,NOLINE,MEFLG,IERCNT
        BYTE PL(132),NAME(8)
        DIMENSION IDTA(3)
        PL(1)=32
        IF(ISZ.EQ.2) GOTO 15
        WORD=IDTA(1)
        CALL INSDAT(IPPOS,4)
        RETURN
15      WORD=IDTA(2)
        CALL INSDAT(IPPOS,4)
        WORD=IDTA(1)
        CALL INSDAT(IPPOS+4,4)
        RETURN
        END
        SUBROUTINE NEWPAG
        IMPLICIT INTEGER (A-Z)
C
C PUTS OUT HEADERS AT TOP OF EACH PAGE
C
        INTEGER PASS
        BYTE NAME(8),FF
        COMMON /LST/ LUNIT,PASS,NAME,NOPAGE,NOLINE,MEFLG,IERCNT
        FF="14
        NOPAGE=NOPAGE+1
        NOLINE = 57
        IF(NOPAGE.EQ.1) FF = 0
        WRITE(LUNIT,10)FF,NAME,NOPAGE
10      FORMAT(' ',1A1,8A1,T28,'M68000 CROSS-ASSEMBLER X1.0
     +',T83,'PAGE ',I3,/)
        RETURN
        END
        SUBROUTINE PAGCHK
        IMPLICIT INTEGER (A-Z)
C
C CHECKS TO SEE IF A PAGE HAS BEEN FILLED
C
        BYTE NAME(8)
        COMMON /LST/ LUNIT,PASS,NAME,NOPAGE,NOLINE,MEFLG,IERCNT
        IF(NOLINE.EQ.0) CALL NEWPAG
        RETURN
        END
        SUBROUTINE ERROR(IERR)
        IMPLICIT INTEGER(A-Z)
C
C-      AND PRINTS ERROR MESSAGE DURING PASS 2
C
        COMMON /LST/ LUNIT,PASS,NAME,NOPAGE,NOLINE,MEFLG,IERCNT
        COMMON /SRC/ LNELEN,ISERR,NOCARD,SRCLNE
        COMMON /PRSE/ OPPTR,MODPTR,OPNPTR,LABEL,CMTPTR
     +,PRFLG,SCANPT,OPCLEN,OPNPT2,IMODE
        COMMON /OBJOUT/ OBJBUF,OBJWC,LFLG,RFLG,DBFLG
        DIMENSION OBJBUF(40)
        COMMON /SYMT/STIND,SYMADR,PC,NOSYM,NEWPC,SYMFLG
        INTEGER*4 PC,NEWPC,SYMADR(512)
        LOGICAL*1 SYMFLG(513),ERRPTR(80),NAME(8),SRCLNE(81)
        LOGICAL*1 LABEL(8)
C
C....   ERRORS ARE IGNORED DURING THE FIRST PASS
C
        IF(PASS.EQ.1) RETURN
C
        PRFLG  = 3
C
C....   WE NEED AT LEAST THREE LINES TO PRINT AN BAD LINE
C
        IF(NOLINE.LE.2) NOLINE = 0
        CALL PAGCHK
C
C....   IF THIS IS NOT THE FIRST ERROR THEN DON'T PRINT THE LINE
C
        IF (MEFLG.EQ.1) GOTO 15
        WRITE(LUNIT,10) NOCARD,(SRCLNE(I),I=1,LNELEN-1)
C
        LSWRDS = OBJWC
        IF(OBJWC.GT.5) LSWRDS=5
C
C       CHECK IF WE HAVE TO GO TO NEXT PAGE
C
        CALL PAGCHK
C
C

        IF(CMTPTR.NE.1)GOTO 80
        OPPTR=1
        GOTO 220
80      GOTO (200,200,200,410,500,600,200,200,200,400),PRFLG+1
200     CALL IHX(2,PC,7)
C
C
        IF(LSWRDS.EQ.0) GOTO 212
205     DO 210,I=1,LSWRDS
210     CALL IHX(1,OBJBUF(I),11+(5*I))
C
C
212     IF(LABEL(1).EQ.0) GOTO 220
        DO 215,I=1,8
215     PL(I+40)=LABEL(I)
220     J=0
        DO 230 I=OPPTR,LNELEN
        PL(J+50)=SRCLNE(I)
        IF(SRCLNE(I).EQ."40) GOTO 240
230     J=J+1
        GOTO 1000
240     III=0
        DO 250 II=I+1,LNELEN
        IF (II.EQ.CMTPTR) III = 25
        PL(57+III)=SRCLNE(II)
        III = III + 1
250     IF ((III + 57).GT.132) GOTO 255
        GOTO 1000
255     PL(132) = 0
        GOTO 1000
C
C       PRFLG = 3  (NEW PAGE)
C
400     CALL NEWPAG
410     RETURN
C
C
500     GOTO 205
C
C
600     GOTO 220
C
C
700     CALL IHX(2,OBJBUF(2),16)
        GOTO 212
C
C
1000    DO 1001 I=48,132
```

```
      SUBROUTINE PST
C
C SORT AND PRINT SYMBOL TABLE
C
      INTEGER PASS,STIND,SYMLIN(512)
      INTEGER*4 PC,NEWPC,SYMADR(512)
      BYTE NAME(8),SYMSYM(8,512),SYMFLG(513),PL(132)
      COMMON/LST/LUNIT,PASS,NAME,NOPAGE,NOLINE,MEFLG,IERCNT
      COMMON/SYMN/SYMSYM,SYMLIN
      COMMON/SYMT/STIND,SYMADR,PC,NOSYM,NEWPC,SYMFLG
      COMMON /CNVT/ WORD,PL
      IF(NOSYM.EQ.0) RETURN
C
C     START OUT WITH CLEAN BUFFER
C
      DO 50 I = 1,132
50    PL(I) = '40
C
C GOTO TOP OF PAGE
C
      CALL NEWPAG
C
C     GENERATE THE SYMBOL LIST A LINE AT A TIME
C
      DO 300 I = 1,NOSYM,5
      DO 210 IDX=0,4
      IF (I+IDX.GT.NOSYM) GOTO 290
      DO 170 IPT=1,7,2
      PL(IPT+(IDX*24)+1) = SYMSYM(IPT,(I+IDX))
170   PL(IPT+(IDX*24)) = SYMSYM(IPT+1,(I+IDX))
      CALL IHX(2,SYMADR(I+IDX),(IDX*24)+12)
      IFTMP = SYMFLG(I+IDX)
      IF ((IFTMP.AND.16).NE.16 ) GOTO 180
      PL((IDX*24)+19) = 'E'
      PL((IDX*24)+20) = 'Q'
180   IF ((IFTMP.AND.8).NE.8  ) GOTO 190
      PL((IDX*24)+19) = 'M'
      PL((IDX*24)+20) = 'U'
190   IF ((IFTMP.AND.1).NE.1) GOTO 200
      PL((IDX*24)+19) = 'U'
      PL((IDX*24)+20) = 'N'
200   IF ((IFTMP.AND.'31).NE.0) GOTO 210
      PL((IDX*24)+19) = ' '
      PL((IDX*24)+20) = ' '
210   CONTINUE
290   WRITE (LUNIT,400) (PL(N),N=1,IDX*24)
      NOLINE = NOLINE -1
      CALL PAGCHK
300   CONTINUE
400   FORMAT (' ',132A1)
      WRITE (LUNIT,410) NOSYM,IERCNT
410   FORMAT (/,' ',I3,' SYMBOLS , ',I3,' ERRORS DETECTED')
      IF (LUNIT.EQ.5) RETURN
      WRITE (5,410)     NOSYM,IERCNT
      RETURN
      END
```

(Continued on page 18, column 2)

```
10    FORMAT(' ',/,' ',I4,35X,80A1:)
      NOLINE = NOLINE - 2
15    DO 20,I=1,SCANPT
20    ERRPTR(I)='40
      ERRPTR(I)='136
      WRITE(LUNIT,30) IERR,(ERRPTR(I),I=1,SCANPT+1)
30    FORMAT(' ++++  ERROR    ',I3,20X,80A1:)
      NOLINE = NOLINE - 1
      IERCNT = IERCNT + 1
      MEFLG = 1
      RETURN
      END
      SUBROUTINE LSTLNE
      IMPLICIT INTEGER (A-Z)
C
C     BUILD LINE (OR LINES IF DC.B DC.W DC.L)
C     FOR DISPLAY
C
      COMMON /SRC/ LNELEN,ISERR,NOCARD,SRCLNE
C
      COMMON /CNVT/ WORD,PL
C
      COMMON /LST/ LUNIT,PASS,NAME,NOPAGE,NOLINE,MEFLG,IERCNT
C
      COMMON /OBJOUT/ OBJBUF,OBJWC,LFLG,RFLG,DBFLG
C
      COMMON /SYMT/ STIND,SYMADR,PC,NOSYM,NEWPC,SYMFLG
C
      COMMON /PRSE/ OPPTR,MODPTR,OPNPTR,LABEL,CMTPTR
     +,PRFLG,SCANPT,OPCLEN,OPNPT2,IMODE
C
      INTEGER*4 PC,NEWPC,SYMADR(512)
      DIMENSION OBJBUF(40)
      BYTE SYMFLG(513),NAME(8),LABEL(8),SRCLNE(81),PL(132)
      DATA PL/132*'40/
C
C     PRFLG = 0    ERRORS DETECTED (PRINT LINE AS READ)
C             1    NO ERRORS DETECTED (PRINT NORMALLY)
C             2    DC.W  / DC.L DIRECTIVES
C             3    SUPRESS PRINTOUT OF LINE
C             4    DC.B DIRECTIVE
C             5    NAM / END / MON DIRECTIVES
C             6    EQU / SET DIRECTIVES
C             7    ORG / RORG DIRECTIVES
C             8    DS   DIRECTIVE
C             9    PAGE DIRECTIVE
C
C
C
C     IF THIS IS THE FIRST PASS, THEN DONT PRINT ANYTHING
C
      IF (PASS.EQ.1) RETURN
C
C     IF CODE IS LONGER THAN FIVE WORDS THEN
C     ONLY PRINT 5 WORDS OF AN INSTRUCTION
```

(Continued on page 18, column 3)

```
1001  IF(PL(I).EQ.0)GOTO 1002
1002  WRITE(LUNIT,1110) NOCARD,(PL(II),II=6,I-1)
1110  FORMAT(' ',I4,132A1)
      DO 1120 II = 1,I
1120  PL(II) = '40
      NOLINE = NOLINE - 1
      RETURN
      END

      SUBROUTINE BLDOBJ
      IMPLICIT INTEGER (A-Z)
C
C     BUILD OBJ FILE
C
      COMMON /FNAM  / FILNAM,OBJFLG
C
      COMMON /OBJOUT/ OBJBUF,OBJWC,LFLG,RFLG,DBFLG
C
      COMMON /CNVT  / WORD,PL
C
      COMMON /SYMT  / STIND,SYMADR,PC,NOSYM,NEWPC,SYMFLG
C
      COMMON /HEXFLG/ ENDFLG,HEXWC,HEXPC,OLDPC
C
      DIMENSION OBJBUF(40),HEXBUF(8)
      INTEGER*4 PC,NEWPC,SYMADR(512),OLDPC,NEWVAL,HEXPC
      LOGICAL*1 SYMFLG(513),PL(132),FILNAM(12)
C
C     CHECK IF OBJ FILE IS TO BE GENERATED
C
      IF (OBJFLG.EQ.0) RETURN
C
C     CHECK FOR THE END OF ASSEMBLY FLAG
C     IF IT IS SET, WRITE OUT THE BALANCE OF THE OBJ BUFFER
C
      IF (ENDFLG.EQ.0) GOTO 10
      IF (HEXWC.NE.0)  CALL WRTOBJ(HEXPC,HEXWC,HEXBUF)
      RETURN
C
C     CHECK THE CURRENT VALUE OF THE PC WITH THAT OF THE ONE SAVED
C     IF THE TWO ARE NOT EQUAL, THEN WRITE OUT THE BALANCE OF THE
C     OBJ BUFFER AND START AT THE NEW PC VAL
C
10    CALL DBLSGL(PC,PC1,PC2)
      CALL DBLSGL(OLDPC,OLDPC1,OLDPC2)
      IF (PC1.NE.OLDPC1) GOTO 50
      IF (PC2.EQ.OLDPC2) GOTO 75
50    IF (HEXWC.NE.0) CALL WRTOBJ(HEXPC,HEXWC,HEXBUF)
      CALL JMOV(PC,HEXPC)
      CALL JMOV(PC,OLDPC)
C
C     EXTRACT OBJECT WORDS FROM OBJECT BUFFER AND
C     PUT THEM INTO AN INTERNAL BUFFER. IF THE
C     INTERNAL BUFFER IS FULL, THEN OUTPUT THE BUFFER.
C
```

(Continued on next page)

68000 Cross Assembler (XASM)

(Listing continued, text begins on page 12)

Listing One

```
75      I = 1
76      HEXWC = HEXWC + 1
        HEXBUF(HEXWC) = OBJBUF(I)
        IF (HEXWC.NE.8) GOTO 99
C....   OBJECT BUFFER IS FULL, OUTPUT IT TO OBJ FILE
C
        CALL WRTOBJ(HEXPC,HEXWC,HEXBUF)
C
C       CALCULATE NEW STARTING PC FOR HEX BUFFER
C
        N = JICVT(I*2,NEWVAL)
        N = JADD(PC,NEWVAL,HEXPC)
99      I = I + 1
        IF (I.LE.OBJWC) GOTO 76
C
C       CALCULATE WHAT THE NEW PC SHOULD BE BY ADDING
C       THE OBJECT WORD COUNT TO THE CURRENT PC
C
        I = JADD(OLDPC,NEWPC,OLDPC)
        RETURN
        END
        SUBROUTINE WRTOBJ(HEXPC,HEXWC,HEXBUF)
        IMPLICIT INTEGER(A-Z)
C
C       OUTPUT THE CONTENTS OF THE OBJECT BUFFER
C
C       HEXPC = STARTING PC FOR BUFFER
C       HEXWC = NUMBER OF WORDS USED IN BUFFER
C       HEXBUF= 8 WORD OBJECT BUFFER
C
        COMMON /CNVT/ WORD,PL
        LOGICAL*1 PL(132)
        INTEGER*4 HEXPC
        DIMENSION HEXBUF(8)
        DO 10, I = 1,80
10      PL(I) = '40
        CALL IHX(2,HEXPC,1)
        PLIDX = 10
        DO 20,I=1,HEXWC
20      CALL IHX(1,HEXBUF(I),PLIDX+(5*(I-1)))
        WRITE (2,100)(PL(I),I=3,10+(5*HEXWC))
100     FORMAT(' ',80A1)
        HEXWC = 0
        DO 900, I = 1,80
900     PL(I) = '40
        RETURN
        END
        SUBROUTINE PRCESS
C
        COMMON /OPWD  / OPNFLG,OPNWC,OPNWRD
C
        COMMON /LST   / LUNIT,PASS,NAME,NOPAGE,NOLINE,NEFLG,IERCNT
C
        COMMON /OBJOUT/ OBJBUF,OBJWC,LFLG,RFLG,BRFLG
C
        COMMON /SRC   / LNELEN,ISERR,NOCARD,SRCLNE
C
        COMMON /SYMT  / STIND,SYMADR,PC,NOSYM,NEWPC,SYMFLG
C
        COMMON /PRSE  / OPPTR,MODPTR,OPNPTR,LABEL,CNTPTR
     +,PRFLG,SCANPT,OPCLEN,OPNPT2,IMODE
C
        COMMON /OPCPTS/ OPTYP,OPSKEL,OPSK2,OPIDX
C
        INTEGER*4 PC,NEWPC,SYMADR(512),SYMVAL,TMPVAL,J2
        LOGICAL*1 SRCLNE(81),LABEL(8),NAME(8),SYMFLG(513)
        DIMENSION OBJBUF(40),OPNWRD(3)
C
C....   SET UP FLAGS THAT CHANGE EACH TIME THRU
C
        CALL I4CLR(NEWPC)
        J2     = 2
        OP1EA  = 0
        OP2EA  = 0
        OP1DA  = 0
        OP2DA  = 0
        OPNWC = 0
C
C....   DECODE OPCODE
C
        CALL DECOPC
        IF(OPTYP.NE.0) GOTO 10
        CALL ERROR(400)
        RETURN
C
C....   SKIP IF NO OPERANDS
C
10      IF(OPNPTR.EQ.0)GOTO 20
C
C....   DECODE FIRST OPERAND
C
        OP1EA=OPNPTR
        CALL EATYP(OP1EA,OP1DA)
        IF(OPNPT2.EQ.0)GOTO 20
C
C....   DECODE SECOND OPERAND
C
        OP2EA=OPNPT2
        CALL EATYP(OP2EA,OP2DA)
C
C....   CHECK FOR OPERANDS
C
20      IF(OP1EA.EQ.10.OR.OP2EA.EQ.10) GOTO 8500
        IF(OPTYP.EQ.1.OR.OPTYP.EQ.2) GOTO 90
        IF(OPNPTR.NE.0) GOTO 90
        CALL ERROR(401)
        IF(IMODE.NE.3) GOTO 125
        I=JICVT(4,NEWPC)
        I=JMUL(NEWPC,OPNWRD(2),NEWPC)
        GOTO 128
122     I=JMOV(OPNWRD(2),NEWPC)
        GOTO 128
125     I=JICVT(2,NEWPC)
        I=JMUL(NEWPC,OPNWRD(2),NEWPC)
128     I=JMOV(PC,OBJBUF(2))
        I=JMOV(PC,SYMVAL)
        GOTO 7005
C
C       ORG
C
130     IF(LABEL(1).EQ.0) GOTO 132
131     CALL ERROR(402)
        RETURN
C
132     RFLG=1
133     PRFLG=7
        IF(OPNPTR.NE.0) GOTO 134
        CALL I4CLR(NEWPC)
        CALL I4CLR(PC)
        RETURN
134     CALL PROCOP(OPNPTR)
        IF(OPNWC.EQ.7) GOTO 135
        CALL I4CLR(PC)
        I=JADD(NEWPC,OPNWRD(2),NEWPC)
        RETURN
135     CALL ERROR(403)
        RETURN
C
C       END <STARTING ADR>
C
140     ISERR=1
        IF(LABEL(1).NE.0) GOTO 131
        PRFLG=5
        RETURN
C
C       EQU
C
150     IF(LABEL(1).EQ.0) GOTO 131
        PRFLG=6
        IF(OPNPTR.EQ.0) GOTO 8500
        CALL PROCOP(OPNPTR)
        IF(OPNWC.EQ.7) RETURN
        CALL SYMTBL(2,OPNWRD(2),LABEL)
        IF((SYMFLG(STIND).AND.'10).EQ.'10)CALL ERROR(409)
        I=JMOV(OPNWRD(2),SYMADR(STIND))
        SYMFLG(STIND)=SYMFLG(STIND).OR.16
        I=JMOV(SYMADR(STIND),OBJBUF(2))
        RETURN
C
C       RORG
C
160     IF(LABEL(1).NE.0) GOTO 131
        RFLG=0
        GOTO 133
```

```
C       PROCESSES SOURCE LINE AFTER IT HAS BEEN PARSED BY PARSE
C
C INPUT:PARSE OUTPUTS
C
C OUTPUT:
C
C  OBJWC    NUMBER OF WORDS REQUIRED FOR INSTRUCTION
C
C  OBJBUF   TABLE OF WORDS GENERATED
C
C  PRFLG 0  ERRORS DETECTED (PRINT LINE AS READ
C        1  NO ERRORS DETECTED (PRINT NORMALLY)
C        2  DC.W/DC.L DIRECTIVES
C        3  DONT PRINT LINE
C        4  DC.B DIRECTIVE
C        5  NAM/END/MON DIRECTIVES
C        6  EQU/SET DIRECTIVE
C        7  ORG/RORG DIRECTIVE
C        8  DS DIRECTIVE
C        9  PAGE DIRECTIVE
C
C  NEWPC    NEW VALUE FOR PC
C
C
C  OP1EA 0  NOT REG OR IMMEDIATE DATA
C        1  D REG
C        2  A REG
C        3  (AN)
C        4  (AN)+
C        5  -(AN)
C        6    DATA
C        7  SR
C        8  CCR
C        9  USP
C        10 ERROR DETECTED
C
C  IMODE 0  NO SIZE SPECIFIED (DEFAULT IS WORD)
C        1  .B
C        2  .W
C        3  .L
C        4  .S  (SHORT BRANCH)
C
C       ERRORS DEFINED.....
C
C       400     UNDEFINED OPCODE
C       401     OPERAND MISSING FOR OPCODE
C       402     NO ORG SPECIFIED FOR ORG INSTRUCTION
C       403     ERROR IN DC OPN VALUE
C       406     GENERAL ERROR IN DECODING
C       407     UNDEFINED SYMBOL
C       408     ERROR IN SIZE OF Y(Ax,Rx) INDEX
C       409     MULT DEFN SYMBOL
C
        IMPLICIT INTEGER (A-Z)
C
```

(Continued on page 20, column 2)

```
        RETURN
C
C....   DEFAULT SIZE IS ONE WORD FOR INSTRUCTIONS
C
90      OBJWC=1
C
C ++++++++++++++++++++++++++++++++++++++++++++++++
C
C GOTO OPCODE EVALUATION ROUTINES VIA OPTYPE
C
C ++++++++++++++++++++++++++++++++++++++++++++++++
        GOTO(100,200,300,500,400,600,700,800,900,1000
     +,1100,1200,1300,1400,1500,1600,1700,1800,1900,2000,2100),OPTYP
C
C ++++++++++++++++++++++++++++++++++++++++++++++++
C
C       PROCESS PSEUDO OPS
C
C ++++++++++++++++++++++++++++++++++++++++++++++++
C
C....   PSEUDO OPS NORMALLY DON'T GENERATE CODE
C....   THE EXECEPTION BEING 'DC'
C
100     OBJWC=0
        GOTO(110,120,130,140,150,195,150,160,170,180,190),OPIDX
C
C       DC
C
110     PRFLG=2
        IFLG = RFLG
        RFLG = 1
111     CALL PROCOP(OPNPTR)
        IF(OPNWC.EQ.0) GOTO 115
        IF(IMODE.EQ.3) OBJWC = OBJWC+2
        IF(IMODE.NE.3) OBJWC = OBJWC+1
        IF(IMODE.EQ.3) OBJBUF(OBJWC-1) = OPNWRD(3)
        IF(IMODE.EQ.3) OBJBUF(OBJWC  ) = OPNWRD(2)
        IF(IMODE.NE.3) OBJBUF(OBJWC  ) = OPNWRD(2)
        IF(SRCLNE(OPNPTR).NE."54) GOTO 119
        OPNPTR = OPNPTR+1
        GOTO 111
115     CALL ERROR(403)
118     RFLG = IFLG
        GOTO 7000
119     IF(IMODE.NE.1.OR.OPNWRD(2).GE.256) GOTO 118
        OBJBUF(OBJWC)=(OBJBUF(OBJWC)*"400)
        GOTO 118
C
C       DS
C
120     PRFLG=7
        IF(OPNPTR.EQ.0) GOTO 8500
        CALL PROCOP(OPNPTR)
        IF(OPNWC.EQ.7) GOTO 134
        IF(IMODE.EQ.1) GOTO 122
```

(Continued on page 20, column 3)

```
C
C       PAGE
C
170     IF(LABEL(1).NE.0)GOTO 131
        LFLG=0
        PRFLG=9
        RETURN
C
C       LIST
C
180     IF(LABEL(1).NE.0)GOTO 131
        LFLG=1
        PRFLG=3
        RETURN
C
C       NLIST
C
190     IF(LABEL(1).NE.0) GOTO 131
        LFLG=0
        PRFLG=3
        RETURN
C
C       NAM
C
195     IF(LABEL(1).NE.0) GOTO 131
        DO 197 I=1,8
197     NAME(I)="40
        N=1
        DO 196 I=OPNPTR,OPNPTR+7
        NAME(N)=SRCLNE(I)
        N=N+1
196     IF(I.EQ.LNELEN-1) RETURN
        RETURN
C ++++++++++++++++++++++++++++++++++++++++++++++++
C
C       PROCESS INHERENT INSTRUCTIONS..IE NOP
C
C ++++++++++++++++++++++++++++++++++++++++++++++++
200     OBJBUF(1)=OPSKEL
        GOTO 7000
C ++++++++++++++++++++++++++++++++++++++++++++++++
C
C       PROCESS MOVE INSTRUCTION
C       <EA>,<EA>  SR,<EA>  <EA>,CCR  <EA>,SR  USP,An  An,USP
C
C ++++++++++++++++++++++++++++++++++++++++++++++++
C
C....   LOOK FOR OBVIOUS MISTAKES
C
300     IF(OP2EA .EQ.6.OR.OP1EA .EQ.8) GOTO 8500
        IF(OP1EA.EQ.9 .AND.OP2EA.NE.2) GOTO 8500
        IF(OP1EA.NE.2 .AND.OP2EA.EQ.9) GOTO 8500
        IF(OPNPTR.EQ.0.OR.OPNPT2.EQ.0) GOTO 8500
C
C....   SR,<EA> - USP,<EA>
```

(Continued on next page)

68000 Cross Assembler (XASM)

(Listing continued, text begins on page 12)

Listing One

```
C
      IF(OP1EA.EQ.7.OR. OP1EA.EQ.9) GOTO 350
C
C....  OP1EA = 1 THRU 5
C
      IF((OP1EA.GE.1).AND.(OP1EA.LE.5)) GOTO 305
C
C....  PROCESS FIRST OPN HERE IF COMPLEX
C
      CALL PROCOP(OPNPTR)
C
C....  CHECK FOR EA TYPES 7-9
C
303   IF(OP2EA.GT.6) GOTO 340
C
C....  CHECK FOR FIRST OPERAND IMMEDIATE MODE ADDRESSING
C
      IF (OP1EA.NE.6) GOTO 304
C
C....  SKIP MOVQ IF FWD REF SYMBOL
C
      IF(OPNFLG.EQ.1) GOTO 304                    ! CANNOT BE FWD REF SYM
      IF(IMODE .NE.3) GOTO 304                    ! MUST BE .L MODE
      IF(OPNWRD(3).EQ. 0) GOTO 301                ! HI WORD MUST BE ZERO
      IF(OPNWRD(3).EQ.-1) GOTO 301                ! OR MINUS ONE
      GOTO 304
C
C....  CHECK IF VAL WITHIN RANGE FOR MOVEQ (+/- 128)
C....  ALSO CHECK IF DESTINATION IS A DATA REGISTER
C
301   I=ICKVAL(OPNWRD(2))
      IF ((I.EQ.0).AND.(OP2EA.EQ.1)) GOTO 330
C
C....  ADD IN OPCODE SIZE BITS
C
304   OBJBUF(1)=OBJBUF(1).OR.'30000
      IF(IMODE.EQ.1) OBJBUF(1)=(OBJBUF(1)).AND.'17777
      IF(IMODE.EQ.3) OBJBUF(1)=(OBJBUF(1)).AND.'27777
C
C....  MOVE IN NUMBERS FOR 1ST AND 2ND EXT WORDS
C
      OBJWC     = OBJWC+OPNWC
      OBJBUF(2) = OPNWRD(2)
      OBJBUF(1) = OPNWRD(1)
      IF(OPNWC.EQ.2) OBJBUF(2) = OPNWRD(3)
      IF(OPNWC.EQ.2) OBJBUF(3) = OPNWRD(2)
      GOTO 310
C
C....  PROCESS EA TYPES 0-5 FOR FIRST OPN
      GOTO 7000
C
C....  HANDLE STUFF FOR EA'S 0 AND 6
C
349   OBJBUF(1)=OBJBUF(1).OR.OPNWRD(1)
      OBJBUF(2)=OPNWRD(2)
      IF(OPNWC.EQ.2)OBJBUF(2)=OPNWRD(3)
      IF(OPNWC.EQ.2)OBJBUF(3)=OPNWRD(2)
      OBJWC=OBJWC+OPNWC
      GOTO 7000
C
C....  GENERATE MOVE SR,<EA> - USP,AN
C
350   IF (OP1EA.EQ.9) GOTO 355                    ! SR,<EA>
      IF (OP2EA.EQ.2) GOTO 8500                   ! USP,AN
      IF (OP2EA.EQ.0) GOTO 353
      OBJBUF(1) = '40300.OR.OP2DA.OR.((OP2EA-1)*'10)
      GOTO 7000
C
353   CALL PROCOP(OPNPT2)
      OBJBUF(2)=OPNWRD(2)
      IF(OPNWC.EQ.2) OBJBUF(2) = OPNWRD(3)
      IF(OPNWC.EQ.2) OBJBUF(3) = OPNWRD(2)
      OBJWC = OBJWC + OPNWC
      OBJBUF(1) = '43000.OR.OPNWRD(1)
      GOTO 7000
C
355   OBJBUF(1) = '47150.OR.OP2DA
      GOTO 7000
C
C +++++++++++++++++++++++++++++++++++++++++++++++++++
C
C     PROCESS CMP INSTRUCTION
C     <EA>,DN <EA>,AN  DATA,<EA> (AY)+,(AX)+
C
C +++++++++++++++++++++++++++++++++++++++++++++++++++
C
400   IF((OP1EA.EQ.6).AND.(OP2EA.NE.2)) GOTO 460  ! CMPI INSTR
      IF((OP1EA.EQ.5).AND.(OP2EA.EQ.5)) GOTO 480  ! CMPM INSTR
      IF((OP2EA.EQ.1).OR. (OP2EA.EQ.2)) GOTO 410  ! CMP <EA>,DN OR AN
      GOTO 8500                                   ! ALL ELSE ILLEGAL
C
C....  PROCESS <EA>,DN <EA>,AN
C
410   IF(OP2EA.EQ.2.AND.IMODE.EQ.1) GOTO 8500     ! CMPA CANT HAVE .B
      IF(OP2EA.NE.2) GOTO 411
      IF(IMODE.EQ.3) OPSKEL = OPSKEL.OR.'500      ! CMPA.L
      IF(IMODE.NE.3) OPSKEL = OPSKEL.OR.'200      ! CMPA.W
411   IF((OP1EA.EQ.0).OR.(OP1EA.EQ.6)) GOTO 415   ! COMPLEX OPN
C
C....  PROCESS FOR REG OPNS
C
412   OBJBUF(1)=OPSKEL.OR.(OP2DA*'1000).OR.((OP1EA-1)*'10).OR.OP1DA
      GOTO 6000
C
C....  PROCESS FOR COMPLEX 1ST OPNS
C
C
C....  GENERATE DN,<EA>
C
      OPSKEL = OPSKEL.OR.(OP1DA*'1000)
      IF(OP2EA.EQ.0) GOTO 511
      OBJBUF(1) = OPSKEL.OR.((OP2EA-1)*'10).OR.OP2DA
      GOTO 6000
C
511   CALL PROCOP(OPNPT2)
514   OBJBUF(2) = OPNWRD(2)
      IF(OPNWC.EQ.2) OBJBUF(2) = OPNWRD(3)
      IF(OPNWC.EQ.2) OBJBUF(3) = OPNWRD(2)
      OBJWC = OBJWC+OPNWC
      OBJBUF(1) = OPSKEL.OR.OPNWRD(1)
      GOTO 6000
C
C....  GENERATE <EA>,DN
C
520   OPSKEL = OPSKEL.OR.(OP2DA*'1000)
      IF(OP1EA.EQ.0) GOTO 522
521   OBJBUF(1) = OPSKEL.OR.((OP1EA-1)*'10).OR.OP1DA
      GOTO 6000
C
522   CALL PROCOP(OPNPTR)
      GOTO 514
C
C....  GENERATE <EA>,AN
C
525   IF (IMODE.EQ.1) GOTO 8500
      IF (IMODE.EQ.3) OPSKEL = OPSKEL .OR. '500
      IF ((IMODE.EQ.2).OR.(IMODE.EQ.0)) OPSKEL = OPSKEL.OR.'200
      OPSKEL = OPSKEL .OR.(OP2DA*'1000)
      IF((OP1EA.EQ.0).OR.(OP1EA.EQ.6)) GOTO 522
      OBJBUF(1) = OPSKEL.OR.((OP1EA-1)*'10).OR.OP1DA
      GOTO 6000
C
C....  GENERATE xxxI
C
530   IF(OP2EA.GT.6) GOTO 8500
C
C....  EVALUATE IMMEDIATE EXPRESSION
C
      CALL PROCOP(OPNPTR)
C
C....  TRY GENERATING SHORT FORM OF INSTRUCTION
C....  AFTER CHECKING TO SEE IF OPERAND WAS FWD REF
C
      IF(OPNFLG.EQ.1) GOTO 536
      IF(OPNWRD(2).GE.1.AND.OPNWRD(2).LE.8) GOTO 550
C
C....  GENERATE EXTENSION WORDS
C....  LENGTH OF OPERAND DEPENDS ON THE IMODE OF INSTRUCTION
C
536   OBJBUF(2) = OPNWRD(2)
      IF(OPNWC.EQ.2)OBJBUF(2) = OPNWRD(3)
      IF(OPNWC.EQ.2)OBJBUF(3) = OPNWRD(2)
537   OBJWC = OBJWC + OPNWC
```

```
C
305     OBJBUF(1)=(((OP1EA-1)*'10).OR.OP1DA)
C
C....   CHK FOR SIMPLE SECOND OPERANDS
C
310     IF(OP2EA.EQ.0) GOTO 315
C
C....   CHK FOR SR,CCR,USP
C
        IF(OP2EA.GT.6) GOTO 340
        GOTO 320
C
C....   CALCULATE COMPLEX SECOND OPN
C
315     CALL PROCOP(OPNPT2)
        OBJBUF(OBJWC+1)=OPNWRD(2)
        IF(OPNWC.EQ.2) OBJBUF(OBJWC+2)=OPNWRD(2)
        IF(OPNWC.EQ.2) OBJBUF(OBJWC+1)=OPNWRD(3)
        OBJWC=OBJWC+OPNWC
        I=(OPNWRD(1).AND.7)*'10
        J=(OPNWRD(1).AND.'70)/8
        OBJBUF(1)=OBJBUF(1).OR.((I+J)*'100).OR.'30000
        GOTO 325
C
C....   PROCESS EA TYPES 0-5 FOR SECOND OPN
C
320     OBJBUF(1)=OBJBUF(1)+(((OP2EA-1).OR.(OP2DA*'10))*'100).OR.'30000
C
C....   ADD IN SIZE BITS
C
325     IF(IMODE.EQ.1)OBJBUF(1)=OBJBUF(1).AND.'17777
        IF(IMODE.EQ.3)OBJBUF(1)=OBJBUF(1).AND.'27777
        GOTO 7000
C
C....   GEN MOVEQ ALSO CLR SIZE BITS IF SET
C
330     OBJBUF(1) = 0
        OBJBUF(1) = (OPNWRD(2).AND.'377).OR.'70000.OR.(OP2DA*'1000)
        GOTO 7000
C
C....   GENERATE MOVE <EA>,SR - <EA>,CCR - AN,USP
C
340     IF(OP2EA.EQ.7) OBJBUF(1)='43300
        IF(OP2EA.EQ.8) OBJBUF(1)='42300
        IF(OP2EA.NE.9) GOTO 342
        OBJBUF(1) = '47140.OR.OP1DA
        GOTO 7000
C
C....   GET NON-REG EA'S IF 0 OR 6
C
342     IF(OP1EA.EQ.0.OR.OP1EA.EQ.6) GOTO 349
C
C....   ELSE JUST ADD OR IN THE EA AND REG
C
        OBJBUF(1)=OBJBUF(1).OR.OP1DA.OR.((OP1EA-1)*'10)
```

(Continued on page 22, column 2)

```
415     CALL PROCOP(OPNPTR)
        OBJBUF(1) = OPSKEL.OR.(OP2DA*'1000).OR.(OPNWRD(1).AND.'77)
        OBJBUF(2)=OPNWRD(2)
        IF(OPNWC.EQ.2) OBJBUF(2)=OPNWRD(3)
        IF(OPNWC.EQ.2) OBJBUF(3)=OPNWRD(2)
        OBJWC=OBJWC+OPNWC
        GOTO 6000
C
C....   CMPI INSTRUCTION
C....   EVALUATE THE IMMEDIATE PART
C
460     CALL PROCOP(OPNPTR)
        OBJWC = OBJWC + OPNWC
        OBJBUF(2)=OPNWRD(2)
        IF(OPNWC.EQ.2) OBJBUF(2) = OPNWRD(3)     ! PLAY GAMES IF 2 WDS
        IF(OPNWC.EQ.2) OBJBUF(3) = OPNWRD(2)
C
C....   CHECK FOR SIMPLE DESTINATION EA
C
        IF((OP2EA.GT.0).AND.(OP2EA.LT.6)) GOTO 470
        IF(OP2EA.GT.6) GOTO 8500
        CALL PROCOP(OPNPT2)
        OBJBUF(1) = OPSK2.OR.(OPNWRD(1).AND.'77)
        OBJBUF(OBJWC+1) = OPNWRD(2)
        IF (OPNWC.EQ.2) OBJBUF(OBJWC+1) = OPNWRD(3)
        IF (OPNWC.EQ.2) OBJBUF(OBJWC+2) = OPNWRD(2)
        OBJWC = OBJWC+OPNWC
        GOTO 6000
C
C....   SECOND EA IS NOT COMPLEX
C
470     OBJBUF(1) = OPSK2.OR.OP2DA.OR.((OP2EA-1)*'10)
        GOTO 6000
C
C....   CMPM (AY)+,(AX)+
C
480     OBJBUF(1)=OPSKEL+((OP2DA*'1000)+OP1DA)
        GOTO 6000
C
C +++++++++++++++++++++++++++++++++++++++++++++++
C
C       PROCESS ADD,SUB INSTRUCTIONS
C       <EA>,DN <EA>,AN  DN,<EA>  DATA,<EA>
C
C +++++++++++++++++++++++++++++++++++++++++++++++
C
500     IF(OP2EA.EQ.2) GOTO 525          ! ADDA,SUBA
        IF(OP1EA.EQ.6) GOTO 530          ! ADDI,SUBI
        IF(OP1EA.EQ.1.OR.OP2EA.EQ.1)  GOTO 510
        GOTO 8500                        ! ALL OTHERS ILLEGAL
C
C....
C
510     IF(OP2EA.EQ.1) GOTO 520
        OPSKEL = OPSKEL .OR. '400
```

(Continued on page 22, column 3)

```
C
C....   IF DEST THRU REG EVAL IT HERE
C
538     IF(OP2EA.EQ.0) GOTO 540
        OBJBUF(1)=OPSK2.OR.((OP2EA-1)*'10).OR.OP2DA
        GOTO 6000
C
C....   EVAL NON-REG DEST
C
540     CALL PROCOP(OPNPT2)
        OBJWC = OBJWC + OPNWC
        OBJBUF(1) = OPSK2.OR.OPNWRD(1)
        IF(OPNWC.EQ.1) OBJBUF(OBJWC  ) = OPNWRD(2)
        IF(OPNWC.EQ.2) OBJBUF(OBJWC+1) = OPNWRD(3)
        IF(OPNWC.EQ.2) OBJBUF(OBJWC  ) = OPNWRD(2)
        GOTO 6000
C
C....   GENERATE xxxQ
C
550     IF(OPNWRD(2).EQ.8) OPNWRD(2) = 0
        IF(OPSK2.EQ.'2000) OPSK2 = '50400
        IF(OPSK2.EQ.'3000) OPSK2 = '50000
        OPSK2 = OPSK2.OR.(OPNWRD(2)*'1000)
        GOTO 538
C
C +++++++++++++++++++++++++++++++++++++++++++++++
C
C       PROCESS AND,OR INSTRUCTIONS
C       <EA>,DN  DN,<EA>   DATA,<EA>
C
C +++++++++++++++++++++++++++++++++++++++++++++++
C
600     IF(OP1EA.EQ.6) GOTO 610
        IF(OP2EA.NE.1) GOTO 620
C
C....   PROCESS <EA>,DN
C
        OPSKEL=OPSKEL+(OP2DA*'1000)
        IF(OP1EA.EQ.0) GOTO 605
        OBJBUF(1)=OPSKEL.OR.OP1DA.OR.((OP1EA-1)*'10)
        GOTO 6000
C
605     CALL PROCOP(OPNPTR)
        OBJBUF(1)=OPSKEL.OR.OPNWRD(1)
        OBJBUF(2)=OPNWRD(2)
        IF(OPNWC.EQ.2) OBJBUF(2)=OPNWRD(3)
        IF(OPNWC.EQ.2) OBJBUF(3)=OPNWRD(2)
        OBJWC=OBJWC+OPNWC
        GOTO 6000
C
C....   PROCESS  DATA,<EA>
C
610     OPSKEL = OPSK2
        IF(OP2EA.EQ.6) GOTO 8500
        CALL PROCOP(OPNPTR)
```

(Continued on next page)

68000 Cross Assembler (XASM)

(Listing continued, text begins on page 12)

Listing One

```
        OBJBUF(2)=OPNWRD(2)
        IF(OPNWC.EQ.2) OBJBUF(2)=OPNWRD(3)
        IF(OPNWC.EQ.2) OBJBUF(3)=OPNWRD(2)
        OBJWC=OBJWC+OPNWC
C
C....   NOW THAT WE HAVE IMMEDIATE DATA GET ,<EA>
C
        IF(OP2EA.EQ.0.AND.OP1EA.EQ.1) GOTO 6000
        IF(OP2EA.EQ.0) GOTO 615
C
C....   CHECK FOR  DATA,SR OR  DATA,CCR
C
        IF(OP2EA.LT.7) GOTO 612
        IF(OP2EA.GT.8) GOTO 8500
        IF((IMODE.EQ.1).AND.(OP2EA.EQ.8)) GOTO 611
        IF((IMODE.EQ.1).OR.(IMODE.EQ.3))  GOTO 8500
611     OBJBUF(1) = OPSKEL.OR.'74
        GOTO 6000
612     OBJBUF(1) = OPSKEL.OR.((OP2EA-1)*'10).OR.OP2DA
        GOTO 6000
C
C....   EVALUATE ,<EA> FOR COMPLEX ADR
C
615     CALL PROCOP(OPNPT2)
630     OBJBUF(OBJWC+1)=OPNWRD(2)
        IF(OPNWC.EQ.2) OBJBUF(OBJWC+1)=OPNWRD(3)
        IF(OPNWC.EQ.2) OBJBUF(OBJWC+2)=OPNWRD(2)
        OBJWC=OBJWC+OPNWC
        OBJBUF(1)=OBJBUF(1).OR.OPSKEL
        GOTO 6000
C
C....   EVALUATE DN,<EA>
C
620     OPSKEL=OPSKEL+(OP1DA*'1000).OR.'400
        IF(OP2EA.EQ.0) GOTO 615
        OBJBUF(1) = OPSKEL.OR.OP2DA.OR.((OP2EA-1)*'10)
        GOTO 6000
C
C ++++++++++++++++++++++++++++++++++++++++++++++++
C
C       PROCESS EOR INSTRUCTION
C       DN,<EA>   DATA,<EA>
C
C ++++++++++++++++++++++++++++++++++++++++++++++++
700     IF(OP1EA.EQ.6) GOTO 610
        IF(OP1EA.NE.1) GOTO 8500
        IF(OP2EA.EQ.0) GOTO 620
        OBJBUF(1)=OPSKEL+((OP1EA-1)*'1000)+OP2DA+((OP1EA-1)*'10)
```

(Continued on column 2)

```
        GOTO 6000
C
C ++++++++++++++++++++++++++++++++++++++++++++++++
C
C       PROCESS ROTATES AND SHIFTS
C       DX,DY  DATA,DY  <EA>
C
C ++++++++++++++++++++++++++++++++++++++++++++++++
800     IF(OP1EA.EQ.1.AND.OP2EA.EQ.1) GOTO 810
        IF(OP1EA.EQ.6.AND.OP2EA.EQ.1) GOTO 820
        IF(OP1EA.EQ.0.AND.OP2EA.EQ.1) GOTO 820
C
C....   PROCESS  <EA>
C
        IF(OP1EA.EQ.0) GOTO 801
        IF(OP1EA.LT.3.OR.OP1EA.GT.5) GOTO 8500
        OBJBUF(1)=OPSKEL+((OP1EA-1)*'10)+OP1DA
        GOTO 7000
C
801     CALL PROCOP(OPNPTR)
        OBJBUF(1)=OPSKEL+OPNWRD(1)
        OBJBUF(2) = OPNWRD(2)
        IF (OPNWC.EQ.2) OBJBUF(2) = OPNWRD(3)
        IF (OPNWC.EQ.2) OBJBUF(3) = OPNWRD(2)
        OBJWC = OBJWC + OPNWC
        GOTO 7000
C
810     OBJBUF(1) = OPSKEL.OR.'40.OR.(OP1DA*'1000).OR.OP2DA
        GOTO 6000
C
820     CALL PROCOP(OPNPTR)
        IF(OPNWRD(2).LT.1.OR.OPNWRD(2).GT.8) GOTO 8500
        IF(OPNWRD(2).EQ.8) OPNWRD(2)=0
        OBJBUF(1)=OPSKEL+(OPNWRD(2)*'1000)+OP2DA
        GOTO 6000
C
C ++++++++++++++++++++++++++++++++++++++++++++++++
C
C
C       PROCESS BRANCH INSTRUCTIONS
C       <LABEL>
C
C ++++++++++++++++++++++++++++++++++++++++++++++++
900     IF(OPNPTR.EQ.0) GOTO 8500
        IF(OP1EA .NE.0) GOTO 8500
        BRFLG = 1
C
C....   GENERATE BRANCH ADDRESS
C
        CALL PROCOP(OPNPTR)
C
C....   CHK FOR FORCED SHORT ADR MODE
C
        IF(IMODE.EQ.4) GOTO 910
C
C....   CHECK FOR FWD REF SYMBOL OR REF BEFORE DEFINITION
```

(Continued on column 3)

```
C
        IF(OPNFLG.EQ.1) GOTO 905
C
C....   CHECK FOR SHORT BRANCH
C
        I = ICKVAL(OPNWRD(2))
        IF((I.EQ.0).AND.(OPNWRD(2).NE.'177600)) GOTO 910
        IF(IMODE.EQ.4) CALL ERROR(404)
C
C....   ELSE GENERATE TWO WORD BRANCH
C
905     OBJBUF(1) = OPSKEL
        OBJBUF(2) = OPNWRD(2)
        OBJWC = 2
        GOTO 920
C
C....   GENERATE SHORT BRANCH
C
910     OBJWC =1
        OPSKEL=OPSKEL+(OPNWRD(2).AND.'377)
        OBJBUF(1) = OPSKEL
920     BRFLG = 0
        GOTO 7000
C ++++++++++++++++++++++++++++++++++++++++++++++++
C
C       PROCESS BIT MODIFICATION INSTRUCTIONS
C       DN,<EA>   DATA,<EA>
C
C ++++++++++++++++++++++++++++++++++++++++++++++++
1000    IF(OP1EA.EQ.1.OR.OP1EA.EQ.6) GOTO 1010
        GOTO 8500
1010    IF(OP1EA.EQ.6) GOTO 1020
        IF(OP2EA.EQ.0) GOTO 1015
C
C....   SIMPLE EA'S
C
        OBJBUF(1) = OPSKEL.OR.(OP1DA*'1000).OR.OP2DA
        OBJBUF(1) = OBJBUF(1) .OR. ((OP2EA-1)*'10)
        GOTO 7000
C
1015    CALL PROCOP(OPNPT2)
        OBJBUF(2) = OPNWRD(2)
        IF (OPNWC.EQ.2) OBJBUF(2) = OPNWRD(3)
        IF (OPNWC.EQ.2) OBJBUF(3) = OPNWRD(2)
        OBJWC = OBJWC + OPNWC
        OBJBUF(1) = OPSKEL.OR.OPNWRD(1).OR.(OP1DA*'1000)
        GOTO 7000
C
1020    CALL PROCOP(OPNPT2)
        IF(OPNWRD(3).NE.0) GOTO 8500
        OBJBUF(2)=OPNWRD(2)
```

(Continued next month.)

The Portable Pidgin

Z80 Macro-Assembly Implementation

When in doubt, write in a machine-independent language. Anyone who has seen a cherished program lie fallow for lack of portability will recognize the wisdom of this maxim. Yet machine-independent languages are normally high-level affairs ill-suited for supporting the compact code and rapid execution times required of many systems programs. Whence my excitement upon learning of William Gale's *Pidgin*, a low-level, structured, portable programming language (*DDJ* #57). I suspect that Dr. Gale's compiler generator Meta4 (*DDJ* #58) is only the first of many powerful systems programs that will be written in the flexible format of Pidgin.

by Herbert Gintis

Dr. Herbert Gintis, Dept. of Economics, Harvard University, Cambridge, MA 02138.

Pidgin is portable because it was designed to be supported by a portable compiler. Indeed, Dr. Gale has provided us with Tincmp, a Pidgin compiler written in Pidgin itself, yet organized to facilitate its relatively easy implementation on any machine. But only relatively! What I shall suggest here is that Tincmp can be implemented with little difficulty on any machine for which a reasonably powerful macro-assembler exists. The example presented here uses a Z80 microprocessor running under CP/M, and the macroprocessor is Venus, a structured editor-assembler-linker-symbolic debugger which I've developed over the past year.

The strategy for implementing Tincmp is straightforward: generate an assembly-language version of the compiler by translating each and every Pidgin statement into a macro.

The result of this exercise is a new Tincmp source file which I have called Tinsource, shown in Listing 2, and a source file of macros called Tinmacro, shown in Listing 1. I shall discuss these two source files in turn.

Since Venus uses standard assembly-language pseudo-ops (e.g., CP/M's ASM.-COM), Tinsource should be easy to understand, subject to the following notes. First, Venus uses the pseudo-op INCBG <filename.filetype> to instruct the assembler to include the specified file at the beginning of the source code before assembly. In this case, Tinmacro will be so included. Second, a string of one to five alphanumerics beginning with a letter, occurring where Venus expects a mnemonic opcode or pseudo-op, and followed by an exclamation point (!), is treated as a macro call.

The meaning of the macros is either the same as that of the corresponding Pidgin statement, is indicated in the remarks, or can be inferred by comparing Tinsource with Tincmp. I have attempted to use obvious mnemonics where possible. Thus "B" stands for "byte", "I" for "int", "E" for "set equal to", "O" for "or", "A" for "and", "EE" for "is equal to", "LE" for "is less than or equal to", and so forth.

Since Venus is a structured assembler, I in fact implemented the control statements WHILE-ON-ENDWHILE and IF-ELSE-ENDIF directly. However, since structured facilities are rare in assemblers, I have rewritten Tinsource by treating these control statements as program labels. Thus a WHILE becomes the string "WH" followed by a number, and ENDWHILE becomes "NDWH" followed by a number. The ON in the WHILE-ENDWHILE construction is translated into the macro IFNOT, which moves beyond the ENDWHILE if the condition tested is false. The IF-ELSE-ENDIF is similarly treated, using the common beginning string IF followed by a number. The CHOOSE ON-CASE construction is directly implemented as a macro.

Tinmacro, given in Listing 1 is also by and large self-explanatory. The macroprocessor of Venus is fairly standard. The name of the macro is written in the label column followed by the pseudo-op MACRO, followed in turn by a list of the prototype variables separated by commas. A proto-type variable is preceded by the ampersand (&). The period (.) is a non-printed string terminator.

Thus if the macroprocessor encounters the line

LD HL, (I.&DEST)

and the current value of &DEST is GG, the line will be expanded as

LD HL, (IGG)

Tinmacro is organized according to the principle that each byte variable in the object code generated (which is in this case an assembly source file) is preceded by the string BB, and each integer variable is preceded by the single letter I. By comparing the listing of Tinmacro with the 8080A macros suggested by Andrew Bender (*DDJ* #65 – March 1982), the reader will notice that I have streamlined some of the coding, using the fact that the Z80 accumulator can be loaded without altering the flag conditions.

To implement Tincmp for your machine, alter Tinmacro and Tinsource to meet the conventions of your assembler. If you do not have a pseudo-op like INCBG, then Tinmacro will have to be merged directly with Tinsource, an easy task on a text editor. If you do not have a macroprocessor, you are probably out of luck, since the full source code of Tinsource with all macros expanded would probably run some 4000 lines.

When Tinsource assembles without errors, then you must implement the Tincmp macros, following the example of Bender's, perhaps with the streamlining suggested in Listing 1. Note that Bender's implementation of the structured control statements uses multiple ORG's, which are disliked by some relocating assemblers. Venus does not mind such multiple ORG's, but there is an easy alternative strategy which I have successfully tested anyway.

The problem arises with the CHOOSE ON $$ and CASE $$ statements in Pidgin. Unlike the C language, Pidgin does not allow a program to "fall through" to the next CASE upon the execution of one CASE, but rather demands exiting the control structure. Thus in generating the assembly source code for the CASE $$ statement, the first line must be a jump over any remaining CASE $$ statements. However, this jump must be suppressed in the *first* such statement after the CHOOSE ON $$. The strategy for handling this is to have the CHOOSE ON $$ place a comment symbol (;) on the top of the stack, and have each CASE $$ immediately place the top of the stack in the destination file, and replace it with a blank ( ), which will be ignored by the assembler. The relevant Tincmp macros will then be:

```
:CHOOSE ON $$;
LD A, (BB↑P1C↑P2C) ;
LD HL,ERASER;
LD (HL),A↑U0S↑L@;S↑U1S;
:CASE $$;
↑!0P↑!9P↑!8P;
↑P9CJP NDCH↑P0N;
CS↑P8N LD A, (BB↑P1C↑P2C) ;
CP (HL) ;
JP NZ,CS↑U1S↑S8N↑L@ S↑P0S;
```

The final step in implementing Tincmp involves developing the system utilities, including the algebraic routines ICOMP, ISUB, CDEHL, IMUL, and IDIV, as well as the CP/M input-output routines START, BREAD, BWRITE, PRT, IOPEN, MSG, CONOUT, and BCLOSE. The requirements for the algebraic routines are well-described in Bender's excellent article. CONOUT is essentially CP/M's Direct Consol I/O (Function 6), but preserving all registers. PRT uses CONOUT to output the string pointed to by the HL register; the string must terminate with a carriage return (0dh in ASCII). MSG is the same as PRT, except that precisely 9 characters are transmitted. The I/O routines must be written using the system utilities associated with your assembler. Their functions, will be obvious from the places in Listing 1 where they are used, after a careful reading of the discussion of these routines in Bender's article.

DDJ

(Listing begins at right)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 514**

Tinmacro (Text begins on page 25)

Listing One

```
(A:TINMACRO)li 11,f
*
* *****          TINMACRO             *****
*
****************************************
*                                      *
*               Listing 1              *
*                                      *
****************************************

*
* Macros for the implementation of TINSOURCE
*
IIEI            MACRO   &A,&B,&C   ;I$$(I$$)=I$$
                LD      HL,(I.&B)
                LD      DE,I.&A
                ADD     HL,HL
                ADD     HL,DE
                EX      DE.HL
                LD      HL,(I.&C)
                LD      A,L
                LD      (DE),A
                INC     DE
                LD      A,H
                LD      (DE),A
                MEND
IINC            MACRO   &A         ;I$$++
                LD      HL,(I.&A)
                INC     HL
                LD      (I.&A),HL
                MEND
BIEB            MACRO   &A,&B,&C   ;$$(I$$)=$$
                LD      HL,(I.&B)
                LD      DE,BB.&A
                ADD     HL,DE
                LD      A,(BB.&C)
                LD      (HL),A
                MEND
BINC            MACRO   &A         ;$$++
                LD      HL,BB.&A
                INC     (HL)
                MEND
BBEB            MACRO   &A,&B,&C   ; $$($$)=$$
                LD      HL,(BB.&B)
                LD      H,0
                LD      DE,BB.&A
                ADD     HL,DE
                LD      A,(BB.&C)
                LD      (HL),A
                MEND
BEBB            MACRO   &A,&B,&C   ;$$=$$($$)
                LD      HL,(BB.&C)
                LD      H,0
                LD      DE,BB.&B
                ADD     HL,DE
                LD      A,(HL)
                LD      (BB.&A),A
                MEND
IEIPI           MACRO   &A,&B,&C   ;I$$=I$$+I$$
                LD      HL,(I.&B)
```

(Continued on column 2)

```
                LD      DE,(I.&C)
                ADD     HL,DE
                LD      (I.&A),HL
                MEND
IEII            MACRO   &A,&B,&C   ;I$$=I$$(I$$)
                LD      HL,(I.&C)
                LD      DE,I.&B
                ADD     HL,HL
                ADD     HL,DE
                LD      E,(HL)
                INC     HL
                LD      D,(HL)
                EX      DE,HL
                LD      (I.&A),HL
                MEND
BEBI            MACRO   &A,&B,&C   ;$$=$$(I$$)
                LD      HL,(I.&C)
                LD      DE,BB.&B
                ADD     HL,DE
                LD      A,(HL)
                LD      (BB.&A),A
                MEND
BEB             MACRO   &A,&B      ;$$=$$
                LD      A,(BB.&B)
                LD      (BB.&A),A
                MEND
IEI             MACRO   &A,&B      ;I$$=I$$
                LD      HL,(I.&B)
                LD      (I.&A),HL
                MEND
BECON           MACRO   &A,&B      ;$$=+$$$
                LD      A,&B
                LD      (BB.&A),A
                MEND
IECON           MACRO   &A,&B      ;I$$=+$$$$$
                LD      HL,&B
                LD      (I.&A),HL
                MEND
BEBPB           MACRO   &A,&B,&C   ;$$=$$+$$
                LD      HL,BB.&C
                LD      A,(BB.&B)
                 ADD     A,(HL)
                 LD      (BB.&A),A
                 MEND
BEBMB            MACRO   &A,&B,&C   ;$$=$$-$$
                 LD      HL,BB.&C
                 LD      A,(BB.&B)
                 SUB     (HL)
                 LD      (BB.&A),A
                 MEND
BEBAB            MACRO   &A,&B,&C   ;$$=$$&$$
                 LD      HL,BB.&C
                 LD      A,(BB.&B)
                 AND     (HL)
                 LD      (BB.&A),A
                 MEND
BEBOB            MACRO   &A,&B,&C   ;$$=$$?$$
                 LD      HL,BB.&C
                 LD      A,(BB.&B)
                 OR      (HL)
                 LD      (BB.&A),A
                 MEND
IEMI             MACRO   &A,&B      ;I$$=-I$$
                 LD      HL,(I.&B)
                 CALL    ICOMP
                 LD      (I.&A),HL
```

(Continued on next page)

Tinmacro (Listing continued, text begins on page 25)

Listing One *(Continued)*

```
ICOMP   EXTRN
        MEND
IEIMI   MACRO   &A,&B,&C    ;I$$=I$$-I$$
        LD      HL,(I.&C)
        EX      DE,HL
        LD      HL,(I.&B)
        CALL    ISUB
        LD      (I.&A),HL
ISUB    EXTRN
        MEND
INEI    MACRO   &BB,&IAA,&IBB    ;$$=I$$!=I$$
        LD      DE,I.&IBB
        LD      HL,I.&IAA
        CALL    INTCK
        LD      (BB.&BB),A
INTCK   EXTRN
        MEND
ILEI    MACRO   &BB,&IAA,&IBB    ;$$=I$$(=I$$
        LD      HL,(I.&IBB)
        EX      DE,HL
        LD      HL,(I.&IAA)
        CALL    CDEHL
CDEHL   EXTRN
        CALL    SETAC
SETAC   EXTRN
        LD      (BB.&BB),A
        MEND
BYTE    MACRO   &BB         ;BYTE $$
        RAM
BB.&BB  DS      1
        REL
        MEND
INT     MACRO   &IAA        ;INT I$$
        RAM
I.&IAA  DS      2
        REL
        MEND
BYTEB   MACRO   &B1,&NNN   ;BYTE $$($$$) OR BYTE $$($$$$$)
        RAM
BB.&B1  DS      &NNN
        REL
        MEND
INTI    MACRO   &IAA,&NNN   ;INT I$$($$$$$)
        RAM
I.&IAA  DS      2 * &NNN
        REL
        MEND
GOSUB   MACRO   &ROUTINE
        CALL    SUB.&ROUTINE
        MEND
STOP    MACRO   &N
        LD      HL,$+6
        CALL    PRT
        JP      0000
        DB      'STOP ON ',&N+'0',:D
        MEND
ISLI    MACRO   &BB,&IAA,&IBB    ;$$=I$$(!I$$
        LD      HL,(I.&IBB)
        EX      DE,HL
        LD      HL,(I.&IAA)
        CALL    CDEHL
CDEHL   EXTRN
        RLA
```

```
        CP      (HL)
        RLA
        AND     1
        LD      (BB.&B1),A
        MEND
IEIB    MACRO   &IAA,&IBB,&B1    ;I$$=I$$($$)
        LD      HL,(BB.&B1)
        LD      H,0
        LD      DE,I.&IBB
        ADD     HL,HL
        ADD     HL,DE
        LD      DE,I.&IAA
        LD      A,(HL)
        LD      (DE),A
        INC     HL
        INC     DE
        LD      A,(HL)
        LD      (DE),A
        MEND
BEI     MACRO   &B1,&IAA    ;$$=I$$
        LD      A,(I.&IAA)
        LD      (BB.&B1),A
        MEND
BLEB    MACRO   &B1,&B2,&B3    ;$$=$$(=$$
        LD      HL,BB.&B3
        LD      A,(BB.&B2)
        CP      (HL)
        CALL    SETAC
        LD      (BB.&B1),A
SETAC   EXTRN
        MEND
BDEC    MACRO   &B1          ;$$--
        LD      HL,BB.&B1
        DEC     (HL)
        MEND
ITMS    MACRO   &IAA,&IBB,&ICC    ;I$$=I$$*I$$
        LD      HL,(I.&ICC)
        EX      DE,HL
        LD      HL,(I.&IBB)
        CALL    IMUL
        LD      (I.&IAA),HL
IMUL    EXTRN
        MEND
IDV     MACRO   &IAA,&IBB,&ICC    ;I$$=I$$/I$$
        LD      HL,(I.&IBB)
        EX      DE,HL
        LD      HL,(I.&ICC)
        CALL    IDIV
        LD      (I.&IAA),HL
IDIV    EXTRN
        MEND
IBEI    MACRO   &IAA,&B1,&ICC    ;I$$($$)=I$$
        LD      HL,(BB.&B1)
        LD      H,0
        ADD     HL,HL
        LD      DE,I.&IAA
        ADD     HL,DE
        LD      DE,I.&ICC
        LD      A,(DE)
        LD      (HL),A
        INC     DE
        INC     HL
        LD      A,(DE)
        LD      (HL),A
        MEND
ITBB    MACRO   &IAA,&B1,&B2    ;UNPACK(I$$,$$,$$)
        LD      HL,(I.&IAA)
        LD      A,L
        LD      (BB.&B1),A
```

```
                AND     1
                LD      (BB.&BB),A
                MEND
RDFIL           MACRO   &CC,&F1     ;READ $$ FROM $$
                LD      HL,BB.&CC
                LD      DE,BB.&F1
                CALL    BREAD
BREAD           EXTRN
                MEND
WRFIL           MACRO   &CC,&F1     ;WRITE $$ INTO $$
                LD      A,(BB.&CC)
                LD      DE,BB.&F1
                CALL    BWRITE
                LD      A,(ER)
                OR      A
                CALL    NZ,PRT
BWRITE          EXTRN
PRT             EXTRN
                MEND
OPEN            MACRO   &F1,&RW,&INM   ;OPEN $$ FOR $$ AT I$$
                LD      DE,BB.&F1
                LD      A,(BB.&RW)
                LD      HL,(I.&INM)
                CALL    IOPEN
                LD      A,(ER)
                OR      A
                CALL    NZ,PRT
PRT             EXTRN
ER              EXTRN
IOPEN           EXTRN
                MEND
MS              MACRO   &STRING     ;PRINT 9 CHARS
                LD      HL,$+5
                CALL    MSG
MSG             EXTRN
                JR      MS.&XX.FF
                DB      &STRING
MS.&XX.FF       REM
                MEND
BNEB            MACRO   &B1,&B2,&B3   ;$$=$$!=$$
                LD      HL,BB.&B3
                LD      A,(BB.&B2)
                CP      (HL)
                LD      A,0
                JR      Z,1
                INC     A
                LD      (BB.&B1),A
                MEND
BEEB            MACRO   &B1,&B2,&B3   ;$$=$$==$$
                LD      HL,BB.&B3
                LD      A,(BB.&B2)
                CP      (HL)
                LD      A,1
                JR      Z,1
                XOR     A
                LD      (BB.&B1),A
                MEND
IEB             MACRO   &IAA,&B1    ;I$$=$$
                LD      A,(BB.&B1)
                LD      L,A
                LD      H,0
                LD      (I.&IAA),HL
                MEND
BSLB            MACRO   &B1,&B2,&B3   ;$$=$$(!$$
                LD      A,(BB.&B2)
                LD      HL,BB.&B3
```

(Continued on top of page 28)

```
                LD      A,H
                LD      (BB.&B2),A
                MEND
IEEI            MACRO   &B1,&IAA,&IBB   ;$$=I$$==I$$
                LD      HL,(I.&IAA)
                LD      DE,(I.&IBB)
                CALL    CDEHL
                LD      A,1
                JR      Z,1
                XOR     A
                LD      (BB.&B1),A
CDEHL           EXTRN
                MEND
GOTO            MACRO   &NN         ;GOTO $$
                JP      LOC.&NN
                MEND
WRITE           MACRO   &B1         ;WRITE $$
                LD      A,(BB.&B1)
                CALL    CONOUT
CONOUT          EXTRN
                MEND
CLOSE           MACRO   &B1         ;CLOSE $$
                LD      HL,BB.&B1
                CALL    BCLOSE
                LD      A,(ER)
                OR      A
                CALL    NZ,PRT
BCLOSE          EXTRN
                MEND
CHSON           MACRO   &B1         ;CHOOSE ON $$
                LD      A,(BB.&B1)
                MEND
IFNOT           MACRO   &B1,&DEST   ;GOTO DEST IF B1 FALSE
                LD      A,(BB.&B1)
                OR      A
                JP      Z,&DEST
                MEND
CASE            MACRO   &B1,&DEST   ; GOTO DEST IF [A] != [&B1]
                LD      HL,BB.&B1
                CP      (HL)
                JP      NZ,&DEST
                MEND
(A:TINMACRO) ff
```

Tinsource

Listing Two

```
(A:TINMACRO) p
(A:TINCMP) li 11,f
*
* *****         TINSOURCE           *****
*
****************************************
*                                      *
*               LISTING 2              *
*                                      *
****************************************
*
* The TINCMP Compiler in Assembly Language
*
                UNLST               ; suppress printer listing
                INCBG   TINMACRO    ; append TINMACRO here
                LIST                ; resume printer listing
```

(Continued on next page)

Tinsource (Listing continued, text begins on page 25)

Listing Two *(Continued)*

```
         XMAC              ; suppress the printing of macro expansion
ER       EXTRN
BBER     EQU      ER       ; all byte variables begin with BB
         BYTE!    AA       ;WORK
         BYTE!    BB       ;WORK
         BYTE!    DD       ;WORK
         BYTE!    EE       ;WORK
         BYTEB!   BF,080   ;EXPANSION BUFFER
         BYTE!    BL       ;BLANK
         BYTE!    BP       ;PTR INTO BF
         BYTE!    C0       ;0
         BYTE!    C1       ;1
         BYTE!    C2       ;2
         BYTE!    C3       ;3
         BYTE!    C4       ;40
         BYTE!    C8       ;80
         BYTE!    C9       ;9
         BYTE!    CC       ;INPUT CHAR
         BYTE!    CX       ;10
*
         BYTE!    DG       ;DIGIT FROM PARAMETER DEF
         BYTEB!   DS,10    ;DIGIT STACK FOR SUB SD
         BYTE!    EF       ;END OF FILE MARK
         BYTEB!   F1,128   ;INPUT BUFFER
         BYTEB!   F2,128   ;OUTPUT BUFFER
         BYTE!    HA       ;'A'
         BYTE!    HF       ;'F'
         BYTE!    LE       ;END OF LIST
         BYTE!    LF       ;LINE FEED CHAR
         BYTEB!   LS,9000  ;LIST OF MACRO DEFS
         BYTE!    MF       ;MACRO REPLACEMENT OPERATOR FLAG
         BYTE!    ML       ;MACRO LENGTH
         BYTE!    MM       ;MIN. MACRO LENGTH
*
         BYTE!    ND       ;# OF DIGITS IN SD OUTPUT
         BYTE!    NL       ;NEW LINE
         BYTE!    O1       ;FETCH CODE
         BYTE!    O2       ;INDEX CODE
         BYTE!    O3       ;DISPOSE CODE
         BYTE!    OA       ;'+' OPERAOR
         BYTE!    OB       ;'!' OPERATOR
         BYTE!    OC       ;'C' CHAR. DISPOSE OPERATOR
         BYTE!    OD       ;'V' DIGIT CONV. FETCH
         BYTE!    OE       ;ESC CHAR
         BYTE!    OG       ;IGNORE CHAR
         BYTE!    OH       ;'H' HEX CONV. FETCH
         BYTE!    OL       ;'L' LITERAL FETCH
         BYTE!    OM       ;'BYTE! ' MULT DISPOSE
         BYTE!    ON       ;'N' NUMERIC LITERAL FETCH
*
         BYTE!    OP       ;'P' PARAMETER FETCH OR DISPOSE
         BYTE!    OR       ;'-' REDUCE (SUBTRACT) DISPOSE
         BYTE!    OS       ;'S' STACK FETCH OR DISPOSE
*
         BYTE!    OT       ;TRACE FLAG
         BYTE!    PP       ;PTR INTO IPR
         BYTE!    RB       ;BEGIN DEF. FLAG
         BYTE!    RC       ;END OF LINE (COMMENT) FLAG
         BYTE!    SF       ;SUBS PARAMETER FLAG
         BYTE!    SP       ;STACK PTR
         BYTE!    TR       ;TRUE IF NO TRACE
         BYTE!    UG       ;USE IGNORE; TRUE UNLESS OG='X'
         BYTE!    UN       ;FLAG FOR NOT SUPRESSING NEW LINESON OUTPUT
         BYTE!    UO       ;USE OPERATIONS; TRUE UNLESS MF='X'
         BEB!     CC,NL
IF32
IF33
         BNEB!    AA,CC,NL     ;AA=CC!=NL
         BNEB!    BB,BP,C8
         BEBAB!   AA,AA,BB
         IFNOT!   AA,NDWH9     ;WHILE < 80 CHARS AND NOT NL
         BBEB!    BF,BP,CC     ;PUT IN BUFFER
         BINC!    BP
         JP       WH9
NDWH9
WH10     BNEB!    AA,CC,NL
         IFNOT!   AA,NDWH10    ;DUMP LONG LINE INPUT
         GOSUB!   GC           ;DUMP LONG LINE INPUT
         JR       WH10
NDWH10
         BBEB!    BF,BP,RC     ;EOL FLAG
         BINC!    BP
         BBEB!    BF,BP,NL
         BEB!     LE,BP        ;END OF LIST
         BLEB!    AA,BP,MM     ;AA=BP<=MM
         IFNOT!   AA,IF34      ;TOO SHORT TO MATCH
         BECON!   ML,0
         GOTO!    17
IF34     BECON!   ML,1
         IEI!     IDP,I00      ;IDP=I00
         BEB!     PP,C0
         IEI!     IJJ,I00
         IEB!     INM,C0       ;INM=C0
WH11     ISLI!    AA,IDP,IED   ;AA=IDP<!IED ;END OF DEFS
         IFNOT!   AA,NDWH11
         BEB!     BP,C0
WH12     BLEB!    AA,BP,LE
         IFNOT!   AA,NDWH12
         BEBI!    AA,LS,IJJ    ;AA=LS(IJJ)
         BEEB!    AA,AA,RC     ;AA=AA==RC
         BEBB!    O3,BF,BP     ;O3=BF(BP)
         BEEB!    O3,O3,RC
         BEBAB!   AA,AA,O3     ;AA=AA&O3
         IFNOT!   AA,IF35
         GOSUB!   DM           ;DO MACRO EXPANSION
         GOTO!    00
IF35     BEBB!    AA,BF,BP
         BEBI!    BB,LS,IJJ
         BEEB!    AA,AA,BB
         IFNOT!   AA,IF36
         GOTO!    01           ;MATCHING
IF36     BNEB!    AA,BB,SF     ;NOT TEMPLATE PARAMETER FLAG
         IFNOT!   AA,IF37
         GOTO!    10           ;MISMATCHED
IF37     BINC!    PP THIS IS PARAMETER
         BEBB!    AA,BF,BP
         IEB!     IAA,AA
         IBEI!    IPR,PP,IAA   ;IPR(PP)=IAA
LOC01    BINC!    BP
         IINC!    IJJ
         JP       WH12
NDWH12
LOC10    BEB!     PP,C0
         IINC!    INM
         IEII!    IDP,ILP,INM  ;IDP=ILP(INM)
         IEI!     IJJ,IDP
         JP       WH11
NDWH11
LOC17    BEB!     BP,C0
WH13     BEBB!    CC,BF,BP     ;CC=BF(BP)
         BEBPB!   O1,BP,C1     ;O1=BP+C1
         BEBB!    AA,BF,O1     ;AA=BF(O1)
         BNEB!    AA,AA,NL
```

```
                BYTE!  UT          ;USE TRACE
                BYTE!  ZR          ;'0'
                INT!   I00         ;0
                INT!   I01         ;1
                INT!   I09         ;9
                INT!   I10         ;10
                INT!   I16         ;16
                INT!   IAA         ;WORK
                INT!   IBB         ;WORK
                INT!   IBC         ;FILE NUMBER
                INT!   IDP         ;DEF PTR WHILE MATCH
                INT!   IED         ;PTR TO END OF DEFS
                INT!   III         ;PTR TO L WHILE READING
                INT!   IJJ         ;PTR TO L
                INT!   ILM         ;MAX LIMIT FOR STORING IN L
                INTI!  ILP,1000    ;PTRS TO MACROS
                INT!   IMP         ;MACRO PTR DURING EXPANSIONS
                INT!   INM         ;# OF MACROS
                INTI!  IPR,10      ;PARAMETER VALUES
                INTI!  ISS,40      ;MAIN STACK
                INT!   ITU         ;VALUE OF PARAMETER TO USE
                INT!   IUU         ;SYMBOL GENERATOR
                INT!   IXX         ;WORK
                INT!   IYY         ;WORK
* BEGINMAIN(AC,IAV)
                CALL   START
START           EXTRN
                BECON! NL,:D
                BECON! LF,:A
                GOSUB! CR
                MS!    'Copyright'
                MS!    ' (C) 1982'
                MS!    ' Herbert '
                MS!    'Gintis   '
                GOSUB! CR
                MS!    'Z80 TINCM'
                MS!    'P Compile'
                MS!    'r CPM Ver'
                MS!    'sion 1.1 '
                GOSUB! CR
                GOSUB! IN          ;INITIALIZE
                GOSUB! RM          ;READ MACRO'S
LOC00
WH7             GOSUB! GI          ;READ CC FROM F1
                BEEB!  AA,ER,C0
                IFNOT! AA,NDWH7
                IFNOT! UG,IF31
WH8             BEEB!  BB,CC,NL    ;BB=CC==NL
                BEEB!  DD,CC,LF
                BEEB!  AA,CC,OG
                BEBOB! EE,BB,DD    ;EE=BB?DD
                BEEB!  BB,CC,BL
                BEBOB! AA,BB,AA
                BEBOB! AA,EE,AA
                IFNOT! AA,NDWH8    ;IGNORE LEADING CHARS (NL,LF,OG,BL)
                GOSUB! GC
                JR     WH8
NDWH8
IF31
                BEB!   BP,C1       ;BUF PTR
                BBEB!  BF,C0,CC    ;BF(C0)=CC
WH9             GOSUB! GC
                BEEB!  AA,CC,NL
                IFNOT! AA,IF32
                GOSUB! GC
                BEEB!  AA,CC,LF
                IFNOT! AA,IF33
```

(Continued on top of page 30)

```
                IFNOT! AA,NDWH13
                IFNOT! ML,IF38
                WRITE! CC
IF38            BEB!   AA,CC
                GOSUB! WA          ;WRITE CC INTO F2
                BINC!  BP
                JP     WH13
NDWH13
                IFNOT! ML,IF39
                GOSUB! CR
IF39            IFNOT! UN,IF40
                BEB!   AA,NL
                GOSUB! WA
                BEB!   AA,LF
                GOSUB! WA
IF40
                JP     WH7
NDWH7
LOC88           MS!    'TINCMP CO'
                MS!    'MPILATION'
                MS!    ' FINISHED'
                GOSUB! CR
                CLOSE! F1
                CLOSE! F2
                JP     :100
*
* Initialize
*
SUBIN           IECON! ILM,8920   ;ILM=+08920
                IECON! I00,0
                IECON! I01,1
                IECON! I10,10
                IECON! I09,9
                BECON! C0,0        ; CO=+000
                BECON! C1,1
                BECON! C2,2
                BECON! C3,3
                BECON! EF,:1A
                BECON! C4,40
                BECON! C8,80
                IECON! I16,16
                BECON! SP,0
                BECON! C9,'9'
                BECON! ZR,'0'
                BECON! BL,' '
                BECON! HF,'F'
                BECON! HA,'A'
                BECON! CX,10
                IEI!   IBC,I01     ;IBC=I01
                BECON! TR,'R'
* CLOSE F1
* ASSOCIATE FCB 1 WITH IBC
                OPEN!  F1,TR,IBC   ;OPEN F1 FOR TR AT IBC
                BECON! TR,'W'      ;TR='W'
                IINC!  IBC         ;IBC++
* CLOSE F2
* ASSOCIATE FCB 2 WITH IBC
                OPEN!  F2,TR,IBC
                GOSUB! GI          ;READ AA FROM F1;'X' SUPRESSES NEW LINE OUTPUT
                BEB!   AA,CC
                BECON! OT,'T'
                BECON! UT,0
                BECON! BB,'X'
                BNEB!  UN,AA,BB    ;UN=AA!=BB;TRUE IF NOT SUPRESS
                GOSUB! GI          ;BEGIN DEF FLAG
                BEB!   RB,CC
                GOSUB! GI          ;COMMEND & EOL FLAG
```

(Continued on next page)

Tinsource (Listing continued, text begins on page 25)

Listing Two *(Continued)*

```
              BEB!    RC,CC
              GOSUB!  GI          ;TEMPLATE PARAMETER FLAG
              BEB!    SF,CC
              GOSUB!  GI          ;EXPANSION OP FLAG
              BEB!    MF,CC
*
* Set UO true iff expansion flag is not 'X'
*
              BECON!  BB,'X'
              BEEB!   AA,MF,BB    ;AA=MF==BB
              IFNOT!  AA,IF1
              BEB!    UO,C0
              JR      IF2
IF1           BEB!    UO,C1
IF2
              BECON!  OP,'P'      ;PARAMETER DESIGNATOR IN OPERATION SEQ
              BECON!  OE,'@'      ;ESC CHAR
              BECON!  OD,'V'      ;CONVERT PARAM
              BECON!  OB,'!'      ;POP STACK DESIG
              BECON!  OS,'S'      ;STACK DESIG
              BECON!  OH,'H'      ;HEX CONST FETCH &WRITE
              BECON!  ON,'N'      ;LITERAL NUMERIC FETCH
              BECON!  OL,'L'      ;LITERAL BYTE FETCH
              BECON!  OC,'C'      ;CHAR OUT DESIG
              BECON!  OA,'+'      ;ADD TO STACK DESIG
              BECON!  OR,'-'      ;SUB FROM STACK
              BECON!  OM,'*'      ;MULTIPLY STACK BY BASE & ADD
              GOSUB!  GI          ;IGNORE CHAR
              BEB!    OG,CC
              BECON!  AA,'X'      ;AA='X'
              BEEB!   BB,AA,OG    ;BB=AA==OG
              IFNOT!  BB,IF3
              BECON!  UG,0
              JR      IF4
IF3           BECON!  UG,1
IF4
              GOSUB!  GI          ;NEW LINE
              BNEB!   AA,NL,CC    ;AA=NL!=CC
              IFNOT!  AA,IF5
              MS!     'FLAG LINE'
              STOP!   1
IF5           IECON!  IUU,100
              RET                 ;ENDSUB IN
*
* Read Macros
*
SUBRM         IEI!    III,I00
              IEB!    INM,C0      ;INM=C0
              BECON!  MM,127      ;MM=127
WH1
              GOSUB!  GI
              BEEB!   AA,ER,C0    ;AA=ER==C0
              IFNOT!  AA,NDWH1
              CHSON!  CC          ;CHOOSE ON CC
              CASE!   OE,CS1      ;ESC CHAR
              GOSUB!  GI
              JP      LOC77
CS1           CASE!   RB,CS2      ;MAC DEF FLAG
              IIEI!   ILP,INM,III   ;ILP(INM)=III
              IINC!   INM
              BECON!  ML,0
              JP      NDCH1
CS2           CASE!   NL,CS2A     ;IGNORE NL
              JP      NDCH1
```

```
              WRITE!  LF
              RET                 ;ENDSUB CR
*
* Do Macro Expansion
*
SUBDM         IEIPI!  IMP,IJJ,I01   ;IMP=IJJ+I01
              IINC!   INM
              IEII!   IDP,ILP,INM   ;IDP=ILP(INM)
WH3           ISLI!   AA,IMP,IDP    ;AA=IMP(!IDP
              IFNOT!  AA,NDWH3
              BEBI!   AA,LS,IMP   ;AA=LS(IMP)
              IFNOT!  UO,IF11
              BEEB!   AA,AA,MF    ;AA=AA==MF
              JR      IF12
IF11          BEB!    AA,C0
IF12
              IFNOT!  AA,IF13
              IINC!   IMP
              BEBI!   O1,LS,IMP   ;O1=LS(IMP)
              IINC!   IMP
              BEBI!   AA,LS,IMP   ;AA=LS(IMP)
              BEB!    O2,AA
              GOSUB!  CD          ;FOR DIGIT CONVERSION
              BEB!    DG,AA
              IINC!   IMP
              BEBI!   O3,LS,IMP
              IFNOT!  UT,IF15
              WRITE!  O1
              WRITE!  O2
              WRITE!  O3
IF15
              CHSON!  O1          ;CHOOSE ON O1
              CASE!   OP,CS4A     ;FETCH PARAMETER
              IEIB!   ITU,IPR,DG  ;ITU=IPR(DG)
              JP      NDCH2
CS4A          CASE!   OD,CS5      ;CONVERT FROM DIGIT TO CHAR
              IEIB!   IAA,IPR,DG  ;IAA=IPR(DG)
              BEI!    AA,IAA      ;AA=IAA
              GOSUB!  CD
              IEB!    ITU,AA      ;ITU=AA
              JP      NDCH2
CS5           CASE!   OB,CS6      ;POP STACK
              IEIB!   ITU,ISS,SP  ;ITU=ISS(SP)
              BLEB!   AA,SP,C0    ;AA=SP(=C0
              IFNOT!  AA,IF16
              MS!     'S STACKER'
              GOSUB!  CR
              BEB!    SP,C1
IF16
              BDEC!   SP
              JP      NDCH2
CS6           CASE!   OS,CS7      ;FETCH TOP STACK, NO POP
              IEIB!   ITU,ISS,SP  ;ITU=ISS(SP)
              JP      NDCH2
CS7           CASE!   OH,CS8      ;FETCH & WRITE HEX CONST. BYTE
              BEB!    AA,O2
              GOSUB!  CH
              IEB!    IAA,AA      ;IAA=AA
              ITMS!   IAA,IAA,I16 IAA=IAA*I16
              BEB!    AA,O3
              GOSUB!  CH
              IEB!    IBB,AA
              IEIPI!  ITU,IAA,IBB
              BEB!    O3,OC
              JP      NDCH2
CS8           CASE!   OL,CS9      ;LITERAL BYTE FETCH
              IEB!    ITU,O2      ;ITU=O2
              JP      NDCH2
CS9           CASE!   ON,CS10     ;LITERAL DIGIT FETCH
```

```
CS2A    CASE!  LF,CS2B     ;IGNORE LF
        JP     NDCH1
CS2B    CASE!  RC,CS3      ;IGNORE COMMENTS
        BIEB!  LS,III,RC   ;LS(III)=RC
        IINC!  III
        BSLB!  AA,ML,MM    ;AA=ML(!MM
        IFNOT! AA,IF6
        BEB!   MM,ML
IF6
WH2     GOSUB! GI
        BNEB!  AA,CC,LF    ;AA=CC!=LF
        IFNOT! AA,NDWH2
        JR     WH2
NDWH2
        JP     NDCH1
CS3     CASE!  OG,CS4      ;IF USING IGNORE, IGNORE
        IFNOT! UG,IF7
        JR     IF8
IF7     JP     LOC77
IF8
        JP     NDCH1
CS4     REM                DEFAULT
LOC77
        BIEB!  LS,III,CC   ;LS(III)=CC
        IINC!  III
        ISLI!  AA,ILM,III  ;AA=ILM(!III
        IFNOT! AA,IF9
        MS!    'MACMEMXST'
        GOSUB! CR
        CLOSE! F1
        STOP!  5
IF9
        BINC!  ML
NDCH1
        JP     WH1
NDWH1
        BNEB!  AA,CC,EF    ;AA=CC!=EF
        IFNOT! AA,IF10
        MS!    'DEFN READ'
        STOP!  2
IF10
        CLOSE! F1
        IECON! IBC,3
* ASSOCIATE FCB 3 WITH IBC
        BECON! TR,'R'
        OPEN!  F1,TR,IBC
        IEI!   IED,III     ;IED=III ;END OF DEFS
        MS!    'LOADED...'
        IEI!   ITU,III
        GOSUB! PN
        MS!    '.BYTES FO'
        MS!    'R DEFINES'
        GOSUB! CR
        IIEI!  ILP,INM,III  ;ILP(INM)=III
        IEI!   ITU,INM
        GOSUB! PN
        MS!    '.MACROS..'
        IEB!   ITU,MM      ;ITU=MM
        GOSUB! PN
        MS!    ' MIN LEN.'
        GOSUB! CR
        RET                ;ENDSUB RM
*
* Do carriage return
*
SUBCR   WRITE! NL
```

(Continued on top of page 32)

```
        BEB!   AA,02
        GOSUB! CD
        IEB!   ITU,AA
        JP     NDCH2
CS10    CASE!  OT,CS11     ;TURN ON TRACE MODE
        BECON! UT,1        ;UT=+001
        JR     NDCH2
CS11    REM                DEFAULT: FETCH UNIQUE #
        IEI!   ITU,IUU     ;ITU=IUU
        IINC!  IUU
NDCH2
        IFNOT! UT,IF17
        IEI!   III,ITU
        GOSUB! PN
        IEIB!  ITU,ISS,SP  ;ITU=ISS(SP)
        GOSUB! PN
        IEB!   ITU,SP      ;ITU=SP
        GOSUB! PN
        IEI!   ITU,III
        GOSUB! CR
IF17
        CHSON! O3          ;CHOOSE ON O3
        CASE!  OC,CS12     ;CHAR OUTPUT
        BEI!   AA,ITU
        GOSUB! WA          ;WRITE AA INTO F2
        JP     NDCH3
CS12    CASE!  OS,CS13     ;PUT ON STACK
        BINC!  SP
        BLEB!  AA,C4,SP
        IFNOT! AA,IF18
        MS!    'S OVERFLO'
        GOSUB! CR
        BEB!   SP,C4
IF18
        IBEI!  ISS,SP,ITU
        JP     NDCH3
CS13    CASE!  OP,CS14 PUT INTO PARAMETER LOCATION
        IBEI!  IPR,DG,ITU
        JP     NDCH3
CS14    CASE!  OA,CS15     ;ADD TO STACK
        IEIB!  IAA,ISS,SP
        IEIPI! IAA,IAA,ITU  ;IAA=IAA+ITU
        IBEI!  ISS,SP,IAA
        JP     NDCH3
CS15    CASE!  OR,CS16     ;SUBTRACT FROM STACK
        IEIB!  IAA,ISS,SP
        IEIMI! IAA,IAA,ITU  ;IAA=IAA-ITU
        IBEI!  ISS,SP,IAA
        JP     NDCH3
CS16    CASE!  OM,CS17     ;MULTIPLY BY BASE AND ADD
        IEIB!  IAA,ISS,SP
        ITMS!  IAA,IAA,I10  ;IAA=IAA*I10
        IEIPI! IAA,IAA,ITU
        IBEI!  ISS,SP,IAA
        JP     NDCH3
CS17    CASE!  OH,CS18     ;OUTPUT HIGH BYTE
        ITBB!  ITU,AA,BB   ;UNPACK(ITU,AA,BB)
        GOSUB! WA          ;WRITE AA INTO F2
        JR     NDCH3
CS18    GOSUB! WN          ;WRITE AS DECIMAL #
NDCH3
        JP     IF14
IF13    REM                ELSE END OF ACTION SECTION
        BEBI!  AA,LS,IMP
        IFNOT! UN,IF19
        BNEB!  BB,AA,RC    ;BB=AA!=RC
```

(Continued on next page)

Tinsource (Listing continued, text begins on page 25)

Listing Two *(Continued)*

```
                JR      IF20
IF19            BEB!    BB,C1
IF20            IFNOT!  BB,IF21
                GOSUB!  WA
                JR      IF22
IF21            BEB!    AA,NL
                GOSUB!  WA
                BEB!    AA,LF
                GOSUB!  WA
IF22
IF14
-               IINC!   IMP
                JP      WH3
NDWH3
                BECON!  UT,0
                RET                  ;ENDSUB DM
*
* Write a byte to output
*
SUBWA           WRFIL!  AA,F2        ;WRITE AA INTO F2
                RET
*
* Get a char from input file
*
SUBGI           RDFIL!  CC,F1        ;READ CC FROM F1
                RET
*
* Get char & goto LOC88 on end
*
SUBGC           GOSUB!  GI           ;GET A CHAR
                BNEB!   AA,ER,C0
                IFNOT!  AA,IF22A
                GOTO!   88
IF22A           BEEB!   AA,CC,EF
                IFNOT!  AA,IF23
                GOTO!   88
IF23
                RET                  ;ENDSUB GC
*
* Converts number to series of ascii digit
*
SUBSD           ISLI!   AA,ITU,I00   ;AA=ITU<'I00
                IFNOT!  AA,IF24
                BECON!  BB,1
                IEMI!   ITU,ITU      ;ITU=-ITU
                JR      IF25
IF24            BECON!  BB,0
IF25
                IEEI!   AA,ITU,I00   ;AA=ITU==I00
                IFNOT!  AA,IF26
                BEB!    ND,C1
                BBEB!   DS,C0,ZR
                JP      IF27
IF26            BEB!    ND,C0
WH4             ISLI!   AA,I00,ITU
                IFNOT!  AA,NDWH4
                IDV!    IYY,ITU,I10  ;IYY=ITU/I10
                ITMS!   IAA,I10,IYY  ;IAA=I10*IYY
                IEIMI!  IXX,ITU,IAA  ;IXX=ITU-IAA
                IEI!    ITU,IYY
                BEI!    AA,IXX
```

(Continued on column 2)

```
                BEBPB!  AA,AA,ZR     ;AA=AA+ZR
                BBEB!   DS,ND,AA     ;DS(ND)=AA
                BINC!   ND
                JR      WH4
NDWH4
IF27
                BBEB!   DS,ND,OR
                BEBPB!  ND,ND,BB     ;MINUS SIGN FOR NEG INTEGER ONLY
                RET                  ;ENDSUB
*
* Write # into F2
*
SUBWN           GOSUB!  SD
WH5             IEB!    IAA,ND       ;WRITE OUT DIGITS
                ISLI!   AA,I00,IAA   ;AA=I00<'IAA
                IFNOT!  AA,NDWH5
                BDEC!   ND
                BEBB!   AA,DS,ND     ;AA=DS(ND)
                GOSUB!  WA
                JR      WH5
NDWH5
                RET                  ;ENDSUB
*
* Write # to terminal
*
SUBPN           GOSUB!  SD
WH6             IEB!    IAA,ND       ;IAA=ND
                ISLI!   AA,I00,IAA
                IFNOT!  AA,NDWH6
                BDEC!   ND
                BEBB!   AA,DS,ND
                WRITE!  AA
                JR      WH6
NDWH6
                WRITE!  BL
                RET                  ;ENDSUB
*
* Convert AA as a decimal digit
*
SUBCD           BLEB!   BB,ZR,AA     ;BB=ZR<=AA
                BLEB!   CC,AA,C9
                BEBAB!  BB,BB,CC     ;BB=BB&CC
                IFNOT!  BB,IF28
                BEBMB!  AA,AA,ZR     ;AA=AA-ZR
                RET                  ;RETURN
IF28            BEB!    AA,C0
                RET                  ;ENDSUB
*
* Convert AA as hex digit
*
SUBCH           BLEB!   BB,ZR,AA
                BLEB!   CC,AA,C9
                BEBAB!  BB,BB,CC
                IFNOT!  BB,IF29
                BEBMB!  AA,AA,ZR
                RET                  ;RETURN
IF29            BLEB!   BB,HA,AA
                BLEB!   CC,AA,HF
                BEBAB!  BB,BB,CC
                IFNOT!  BB,IF30
                BEBMB!  AA,AA,HA
                BEBPB!  AA,AA,CX
                RET                  ;RETURN
IF30            BEB!    AA,C0
                RET                  ;ENDSUB
*
(A:TINCMP)ff
```

End Listing

Simplified 68000 Mnemonics

If the 68000 is to become the most important new computer of the next five years, there will necessarily be a tremendous amount of interest in its assembly language. The profusion of features and facilities on the 68000, however, makes it very difficult to get a clear overall view of the instruction set. If one is to program reasonably well in assembly language, one must be able to lay out in one's mind which facilities are there and which are not. With the 68000, it often seems as if one is let loose upon a tropical island, full of flowers of dazzling beauty but so much underbrush that it seems impossible to scale its central mountain peak and survey the totality of the landscape.

The main problem is with the mnemonics. This is not to say that the mnemonics show any glaring examples of bad design. They are serviceable, as far as they go; it is simply that it is possible to do much better. The purpose of this paper is to *propose* a new collection of mnemonics for the 68000. This is not to say that an assembler has been written, using these mnemonics (at the moment the author does not have access to a 68000-based system). Rather, the purpose here is to show the many advantages of this set of mnemonics over Motorola's. There is precedent for such redesigns; for example, the mnemonics of the UNIVAC 1100 series were redone by CSC in 1962 in connection with CSC's design of the EXEC II operating system for the 1100 series (a project in which this author took part).

The new mnemonics are given in Table I. The notation used in Table I is explained in Table II. It is important to note that the new mnemonics allow the entire instruction repertoire of the 68000 – together with very brief explanations of all instructions – to be presented upon one single-spaced page, with all notational conventions laid out on another single-spaced page.

A program selected at random from a recent book on the 68000[1] is presented in Table III with both the old and new mnemonics. Except for labels, which remain the same in the two cases, the total number of keystrokes of non-blank characters has been reduced from 196 to 98 – a reduction of 50%. The simplicity of understanding for the new mnemonics makes it easy, in fact, to spot a bug in the program of Table III. This will be explained in more detail at the end of the paper.

by W.D. Maurer

W. D. Maurer, George Washington University, S.E.A.S., Washington, DC 20052.

The basic ideas behind the new mnemonics are as follows:

(1) *The data registers and the address registers all receive new designations.* The data registers are now called the A, B, C, D, E, F, G, and H registers, each of which is 32 bits long. The address registers are now called the S, T, U, V, W, X, Y, and Z registers, and each of these is 32 bits long. In particular, the S register is the stack pointer, known conventionally as A7. (The original designations for the registers A through H are D0 through D7, and those for S through Z are A7 through A0, in reverse order, with A0 corresponding to Z.)

(2) *There are no suffixes B, W, and L for byte, word, and long word operations.* As we shall see, most instructions do not need such suffixes because they operate on data which has been defined as byte,

G	OPERATION	DESCRIPTION
3	A v,i	Add i to v
1	Aa v	Add v to a (v ≠ 1t)
	ABab	Add BCD 1b to 1a
	ABtu	Add BCD 1MDu to 1MDt
4	AMa v	Add a to v (memory)
4	ASd v2	Arith. shift v2 by 1
	ASd ma,sc	Arith. shift ma by sc
1	At vx	Add vx to t
	AXmab	Add extended mb to ma
	AXmtu	" " mMDu to mMDt
2	An v	Add n to v (v ≠ 1t)
	B p	Branch to p
	Baf i2	Test bit i2 of a, op f
	Bafb	" " y (in b) " " " "
	BaT i2	Test bit i2 of a
	BaTb	Test bit y (in b) of a
	Bcc p	Branch to p on cc
4	Bf v1,i2	Test bit i2 of v1,op f
4	Bfa v1	" " y (in a) " " " "
	BS p	Branch to subroutine p
5	BT v1,i2	Test bit i2 of v1
5	BTa v1	Test bit y (in a)of v1
3	C v,i	Compare i with v
1	Ca v	Compare a with v(v≠1t)
6	CKa v2	Check 2a bounded by v2
	Cmtu	Compare mMtI with mMuI
1	Ct vx	Compare t with vx
	Dcca p	Bcc β;a←a-1;a=-1?→p;β:
6	DSa v2	Divide signed 4a/v2
6	DUa v2	Divide unsigned 4a/v2
	Ea	Exchange halves of a
	Egh	Exchange 4g with 4h
	FAt -i2	Frame allocation (iMt)
	FDt	Frame deallocation
	H i2	Halt (status reg.←i2)
7	J v	Jump to v
7	JS v	Jump to subroutine v
7	LAt v	Load t with addr. of v
4	LSd v2	Logical shift v2 by 1
	LSd ma,sc	Logical shift ma by sc
1	M v,w	Move w to v (w≠1t)
1	Ma v	Move v to a (v≠1t)
6	MCC v2	Move v2 (byte 2) to CC
	MKa k	Move signed const. k→a
1	MMa v	Move a to v (memory)
1	MMt vx	Move t to vx (memory)
9	MR rs,vx	Move vx → registers rs
	MRxt rs	Move regs(=MR rs,xMtI)
6	MSa v2	Multiply signed 2a*v2
3	MSM v2	Move status reg. to v2
P6	MSR v2	Move v2 to status reg.
P	MSt	Move t to S
1	Mt vx	Move vx to t
P	MtS	Move S to t
6	MUa v2	Multiply uns. 2a*v2
3	N v,i	Logical AND i to v
6	Na v	Logical AND v to a
	NBa	Negate BCD 1a
4	NBM v1	Negate BCD v1 in mem.
	NCC i1	Logical AND i1 to CC
	NGma	Negate ma (2's comp.)
4	NGM v	Negate v in memory
3	NMa v	AND a to v (memory)
	NO	No operation
P	NSR i2	Logical AND i2 to SR
	NTma	NOT ma (ones' compl.)
4	NTM v	NOT v in memory
	NXma	Negate extended ma
4	NXM v	" " v in memory
3	O v,i	Logical OR i to v
6	Oa v	Logical OR v to a
	OCC i1	Logical OR i1 to CC
3	OMa v	OR a to v (memory)
P	OSR i2	Logical OR i2 to SS
7	PA v	Push address of v
1	PH v	Push v (= M mMDS,v)
1	PL v	Pull v (= M v,mMSI)
	PMxa Q(t)	xa←Q(t)alternate bytes
	Pxa Q(t)	Q(t)←xa " " (periph.)
	R	Return from subroutine
	RE	" " excptn. (pull SR)
8	RM rs,vx	Regs to memory rs → vx
	RMxt rs	Regs-mem.(=RM rs,xMDt)
	ROd v2	Rotate v2 by 1
	ROd ma,sc	Rotate ma by sc
	RR	Ret.&restore (pull CC)
	RS	Reset external devices
	RXd v2	Rot. extended v2 by 1
	RXd ma,sc	Rot. extended ma by sc
3	S v,i	Subtract i from v
1	Sa v	Subtr. v from a (v≠1t)
	SBab	Subtr. BCD 1b from 1a
	SBtu	" " 1MDu from 1MDt
3	Scc v1	Set v1=-1 (if cc) or 0
4	SMa v	Subtr. a from v (mem.)
1	St vx	Subtract vx from t
	SXmab	Subtr. extended ma-mb
	SXmtu	Subtr. ext. mMDt-mMDu
2	Sn v	Subtr. n from v (v≠1t)
3	TE v	Test v, set flags N&Z
	TR vn	Trap, using vector vn
3	TS v1	TE v1; v1(hi-order) =1
	TV	BVC β; TR 7; β:
3	X v,i	Exclusive OR i to v
	XCC i1	Exclusive OR i1 to CC
3	XMa v	Excl. OR a to v (mem.)
P	XSR i2	Exclusive OR i2 to SR
	Xxa	Extend sign a (to xa)
3	Z v	Zero (clear) v

Table I

word, or long word data, which determines the type of the operation automatically. In cases where the length must be given, however, it is given in bytes as 1, 2, or 4. This has the immediate advantage of avoiding the non-standard term "word." (A word has 32 bits, not 16, on the IBM 360 and 370, for example.)

The length designation appears in the following places:

(a) *In certain of the mnemonics.* The instruction whose Motorola mnemonic is ADDX.L D4,D3 (Add Extended, Long Word, register D4 to register D3) is simply AX4DE in the new mnemonics. Here D and E denote the D register (formerly D3) and the E register (formerly D4) and the 4 denotes, very simply, "four bytes." Note that the order is "destination first, then source"; see point (3) below to appreciate how much simpler this makes a great number of the mnemonics. The corresponding byte and word comparison instructions ADDX.B D4,D3 and ADDX.W D4, D3 become AX1DE and AX2DE respectively.

```
a = A, B, C, D, E, F, G, or H (data registers)
b = A, B, C, D, E, F, G, or H (data registers)
CC = condition code register (rightmost byte of SR)
cc = one of the following forms:
       T     true                  VC   overflow clear
       F     false                 VS   overflow set
       HI    high                  PL   plus
       LS    low or same           MI   minus
       CC    carry clear           GE   greater or equal
       CS    carry set             LT   less than
       NE    not equal             GT   greater than
       EQ    equal                 LE   less or equal
d = L (left shift) or R (right shift)
e = label, label±j, or other address expression (length defined
     in definition of given label) or mBlabel (length m)
f = N (op=AND with 0), O (op=OR with 1), or X (op=XOR with 1)
g = A, B, C, D, E, F, G, H, S (stack pointer), T, U, V, W,
     X, Y, or Z (data register or address register); length = 4
h -- same as g (second register in exchange instruction)
i = y (constant) or e (address of e) (immediate data)
i1 -- same as i but length must be 1 (immediate data)
i2 -- same as i but length must be 2 (immediate data)
j = $hh... or dd... or %bb... where the h's are hexadecimal digits,
     the d's decimal digits, and the b's binary digits (constant)
k -- +k is +0 through +127; -k is -1 through -128 (constant in move
     instruction MK; 8-bit displacement in based indexed modes)
m -- 1, 2, or 4 (length, or number of bytes in register or memory)
n -- 1, 2, 3, 4, 5, 6, 7, or 8 (additive or subtractive constant)
p = q3, but *-32766 ≤ p ≤ *+32769 (relative address; *=this loc.)
q = e (data) or mBj (data, length m, at address j) (memory loc.)
q2 -- same as q but address must have length 2 (memory location)
q3 -- same as q but address must have length 3 (memory location)
r = q3, but *-126 ≤ r ≤ *+129 (relative address; *=this location)
rs = z1/z2/... where each zk = g or g-h (g, h as above; g-h means
      registers g, g+1, ..., h) (multiple registers for MR and RM)
sc = 1, 2, 3, 4, 5, 6, 7, 8, A, B, C, D, E, F, G, H (shift count
      -- in rightmost 6 bits of register A-H if this is specified)
SR = status register (length 2; CC = rightmost byte of SR)
t = S (stack pointer), T, U, V, W, X, Y, or Z (address registers)
u = S (stack pointer), T, U, V, W, X, Y, or Z (address registers)
v = one of the following forms (variable in register or memory):
     #i (immediate)                    (Groups 1 and 6 only)
     q3 (absolute long)                (All groups)
     q2 (absolute short)               (All groups)
     ma (data register direct)         (Groups 1, 2, 3, and 6 only)
     mt (address register direct)      (Groups 1 and 2 only)
     mMt (address register indirect)   (All groups)
     mMtI (post-increment)             (All groups except 7 and 8)
     mMDt (pre-decrement)              (All groups except 7 and 9)
     mMtg or mMtg±k (based indexed short) (all groups)
     mMt4g or mMt4g±k (based indexed long) (all groups)
     q2(t) or mMt±q2 (based)           (All groups)
     p (relative)                      (Groups 1 and 5 through 9 only)
     r(t) or r(4t) (relative indexed)  (Groups 1 and 5 through 9 only)
vm -- like v but with length m (variable in register or memory)
vn -- 0 through 15 (vector number in trap instruction)
w -- like v in M v,w but group 3 and length must equal length of v
x -- like m, but x ≠ 1 (vx is like v but with length 2 or 4)
y = mBj (length m) or j (length as small as possible) (integer)
```

Table II

(b) *Preceding register designations.* The instruction MOVE.L A3,A6 (old mnemonics) moves A3 to A6. Renaming these registers W and T respectively, we can write M 4T,4W (reversing the order as before). It is simpler, however, to write MT 4W where MT means "move to the T register" and 4W means "four bytes of the W register." The corresponding byte and word move instructions MOVE.B A3,A6 and MOVE.W A3, A6 become MT 1W and MT 2W respectively.

(c) *As part of more general addressing modes.* Address register indirect addressing, involving the X register, in a one-byte instruction, would be denoted by 1MX (*one* byte of *memory* indexed by *X*.) The same thing, with auto-increment or auto-decrement, would be 1MXI or 1MDX respectively. (Note that the I, for "increment," *follows* the register name, whereas D, for "decrement," *precedes* it; this resembles the placement of + and – in the Motorola mnemonics, and is done for the same reason – because incrementation follows the reference to memory, while decrementation precedes it.) Even the most complex mode, namely based indexed long, is straightforward; thus 2MU4E–9 means "*two* bytes of *memory*, indexed by the *U* register and by all *four* bytes of the *E* register, *minus* the displacement *9*." The same thing in Motorola mnemonics would be –9(A5,D4.L) – over 50% more characters (not counting the extra .W on the instruction mnemonic) and lacking the straightforward readability of the new expression.

(3) *Most operation codes have only one argument.* In fact the *only* operation codes with two arguments are moves, shifts, and immediate addressing instructions. Thus M KPREV,KCURR sets KPREV (in memory) equal to KCURR (in memory); A FCOUNT,5 adds 5 to FCOUNT in memory; ASL 1H,4 shifts 1H (*one* byte of the *H* register, formerly known as D7) left arithmetically by 4. (If the mnemonic variations are counted separately – if MA through MH are counted as eight instructions instead of one, for example – then almost 75% of the mnemonics have *no* arguments; about 25% have one; and only 2% have two.)

The destination register in a great number of the arguments is incorporated into the mnemonic. For the C register, for example (formerly D2), we have MC (move to C, AC, SC, CC (add, subtract, and compare), MSC, MUC, DSC, DUC (multiply and divide by C, signed and unsigned), NC (AND), OC (OR), and so on. For a few "operate to memory instructions" such as AMC (add to memory), SMC, NMC, and OMC, the C register is the source, rather than the destination. There are also EC (exchange halves of C), ECD (exchange C and D), NG4C (set C to its two's complement), X2C and X4C (extend sign to two bytes or four bytes of C), and other such exceptional cases.

(4) *All operations on data have lengths determined by the data unless otherwise directed.* If AMOUNT is a 32-bit variable, then MF AMOUNT moves AMOUNT to the F register, and is a long-word (32-bit) operation. To move only one or two bytes of AMOUNT to the F register, one uses MF 1BAMOUNT or MF 2BAMOUNT respectively.

Tables I and II are meant to be used in tandem. As an example, consider the second entry in Table I, namely A*a v* (Add *v* to *a*). By consulting Table II, one learns that *a* may be A, B, C, D, E, F, G, or H, and that *v* may have one of several forms. Let us consider the form *mt*. Here *m* is 1, 2, or 4, but in the description of A*a v* we find the designation "$v \neq 1t$"; that is, *v* cannot be of the form 1*t*, so that *m*, here, must be 2 or 4, while *t* is S, T, U, V, W, X, Y, or Z (again by reference to Table II). Thus there are the following possibilities (for example):

AA 2X
(Add to A from two bytes of X)

AD 4T
(Add to D from four bytes of T)

for a total of 128 (8 x 2 x 8) possibilities. Another form of *v*, in this same instruction, is *q*3, described as "same as *q* but address must have length 3." So this is a 24-bit (or three-byte) address, and we have the following further possibilities for *v*:

$q = e =$ label
(label of location with three-byte address)

$q = m\mathrm{B}j =$ 4B$hh. . .
(length 4, but *address* still has length 3)

$q = e =$ label$\pm j$ = label+dd. . .

$q = e = m$Blabel
(data has length *m*; address has length 3)

This means that if NST is a two-byte variable, ECOUNT a four-byte variable, and F7 an array of single-byte variables, we could also write

AF ECOUNT
(Add to four bytes of F from ECOUNT)

AD 4B$FC
(Add to D from four bytes starting at address $FC)

AG F7+50
(Add to one byte of G from byte 50 of F7)

AA 1BNST
(Add to one byte of A from one byte of NST)

Still another form of *v*, in this same instruction, is #*i*, for immediate data. By Table II, we can have $i = y = m\mathrm{B}j =$ 4B$hh. . . which gives us the possible instruction

AD #4B$FC
(Add $FC to four bytes of D)

Note the two meanings of the 4 in 4B$FC – in this instruction it means "the constant $FC as a four-byte constant," whereas in AD 4B$FC it means "four bytes of memory."

The column headed G in Table I is the *group*. Instructions belong to groups, and each group can use only certain of the *v* options as specified in Table II. For example, the instruction LA*t v* belongs to group 7. This means that *v* can be of

OLD MNEMONICS

```
PGM_9_4A MOVEA.L  LIST,A0
         CLR.W    D0
         MOVE.B   (A0)+,D0
         LEA      -1(A0,D0.W),A1
SORT     CLR.W    D1
         MOVEA.L  A0,A2
NEXT     MOVE.B   (A2)+,D0
         CMP.B    (A2),D0
         BCC.S    NSWITCH
         MOVE.B   (A2),D1
         MOVE.B   D1,-1(A2)
         MOVE.B   D0,(A2)
         ADDQ.W   #1,D1
NSWITCH  CMPA.L   A2,A1
         BHI      NEXT
         TST.W    D1
         BNE      SORT
         RTS
```

NEW MNEMONICS

```
PGM_9_4A MZ    LIST
         Z     2A
         MA    1MZI
         LAY   1MZA-1
SORT     Z     2B
         MX    4Z
NEXT     MA    1MXI
         CA    1MX
         BCC   NSWITCH
         MB    1MX
         MMB   1MX-1
         MMA   1MX
         A1    2B
NSWITCH  CY    4X
         BHI   NEXT
         TE    2B
         BNE   SORT
         R
```

REGISTERS

OLD	D0	D1	D2	D3	D4	D5	D6	D7	A0	A1	A2	A3	A4	A5	A6	A7
NEW	A	B	C	D	E	F	G	H	Z	Y	X	W	V	U	T	S

Table III

the form $r(t)$ or $r(4t)$, since these are specified in Table II as "groups 1 and 5 through 9 only." Thus if WTABLE is a suitable relative address, then each of the following is *legal*:

LAT WTABLE(Z)
(Load address of WTABLE plus Z into T)

LAV WTABLE(4W)
(Load address of WTABLE plus W into V)

where two bytes of Z, and four bytes of W, are used. On the other hand, v cannot be of the form mMtI or mMDt, since these are specified in Table II as "all groups except 7 and 8" or "all groups except 7 and 9." Thus each of the following is *illegal*:

LAT 2MZI
LAV 4MDW

It should be noted that the length (1, 2, or 4) is meaningless in the case of the specific instructions LAt, PAt, J, and JS. Thus in the program of Table III, LAY 1MZA-1 means "*L*oad *A*ddress, into *Y*, of the *memory* word indexed by *Z* and by *A*, *minus 1*." The 1 before the M means nothing, but it must be present (although 2 or 4 would do just as well), since LAY MZA-1 would load the address of a variable called MZA, minus 1. (Privileged instructions are group P.)

Examples of the v options which start with a length (1, 2, or 4) are as follows:

(1) 1C (one byte of the C register; the other three bytes are ignored)
(2) 2V (two bytes of the V register; extended through the other two bytes)
(3) 1MU (one byte in memory, indexed by U – that is, the address of this byte is contained in U)
(4) 2MWI (two bytes in memory, indexed by W, and then increment W by 2 after making the reference)
(5) 4MDT (four bytes in memory, indexed by T, and decrement T by 4 *before* making the reference)
(6) 2MUE (two bytes in memory, indexed by U and by E – that is, the address of this byte is the sum of U and two bytes of E)
(7) 1MWX (one byte in memory, indexed by W and by X)
(8) 4MZ4G (four bytes in memory, indexed by Z and all four bytes of G)
(9) 4MXY+8 (four bytes in memory,

indexed by X and by Y, plus 8 – that is, the address of this byte is the sum of X and two bytes of Y, plus 8)

(10) 1MZ4S-16 (one byte in memory, indexed by Z and by all four bytes of the stack pointer S, minus 16)

Table IV gives examples of all the instructions in both the old and the new mnemonics; Table V gives a more expanded explanation of the program of Table III. We may note from Table V that the use of the new mnemonics makes it much easier to spot a bug in this sorting program, intended to sort a table of unsigned, single-byte quantities into *descending* order. The B register (formerly D1) is being used as the exchange flag: set to zero (Z 2B), incremented to show the exchange (A1 2B), and tested at the end TE 2B). But it is also used as a temporary register in moving 1MX to 1MX-1 (in the instructions MB 1MX and MMB 1MX-1). If the bug is not fixed, the instruction A1 2B might increment -1 to zero, making it look like the exchange flag is still clear.

MNEMONIC	EXAMPLE	MOTOROLA MNEMONIC	MNEMONIC	EXAMPLE	MOTOROLA MNEMONIC
A v,i	A 2MZI,1	ADD.W #1,(A0)+	N v,i	N 4MDY,8	AND.L #8,-(A1)
Aa v	AA 4MDY	ADD.L -(A1),D0	Na v	NB 1MXI	AND.B (A2)+,D1
ABab	ABGH	ABCD D7,D6	NBa	NBC	NBCD D2
ABtu	ABWX	ABCD -(A2),-(A3)	NBM v1	NBM 1MZT	NBCD 0(A0,A6)
AMa v	AMB BYT1	ADD.B D1,BYT1	NCC i1	NCC $1A	AND.B #$1A,SR
ASd v2	ASR WD1	ASR WD1	NGma	NG1A	NEG.B D0
ASd ma,sc	ASL 2C,4	ASL.W 4,D2	NGM v	NGM 2MUD	NEG.W 0(A5,D3)
At vx	AY 4MV	ADD.L (A4),A1	NMa v	NMD 4MDS	AND.L D3,-(A7)
AXmab	AX1DE	ADDX.B D4,D3	NO	NO	NOP
AXmtu	AX2UZ	ADDX.W -(A0),-(A5)	NSR i2	NSR $F0F	AND.W #$F0F,SR
An v	A4 4F	ADDQ.L #4,D5	NTma	NT2E	NOT.W D4
B p	B LABEL1	BRA LABEL1	NTM v	NTM BYT1	NOT.B BYT1
Baf i2	BGN 8	BCLR #8,D6	NXma	NX4F	NEGX.L D5
Bafb	BHOF	BSET D5,D7	NXM v	NXM 1MS	NEGX.B (A7)
BaT i2	BET $10	BTST $10,D4	O v,i	O 2G,$AA	OR.W #$AA,D6
BaTb	BDTC	BTST D2,D3	Oa v	OH BYT1	OR.B BYT1,D7
Bcc p	BEQ BETA	BEQ BETA	OCC i1	OCC 8	OR.B #8,SR
Bf v1,i2	BX M9,6	BCHG 6,M9	OMa v	OMB 4MUI	OR.L D1,(A5)+
Bfa v1	BNB 1MTI	BCLR D1,(A6)+	OSR i2	OSR $200	OR.W #$200,SR
BS p	BS PROG3	BSR PROG3	PA v	PA 1MV	PEA (A4)
BT v1,i2	BT M8,3	BTST 3,M8	PH v	PH WD1	MOVE.W WD1,-(A7)
BTa v1	BTA 1MDZ	BTST D0,-(A0)	PL v	PL 4A	MOVE.L (A7)+,D0
C v,i	C 4MSI,5	CMP.L #5,(A7)+	PMxa Q(t)	PM2B J(Z)	MOVEP.W D1,J(A0)
Ca v	CD 1A	CMP.B D0,D3	Pxa Q(t)	P4C J(Y)	MOVEP.L J(A1),D2
CKa v2	CKB WD1	CHK WD1,D1	R	R	RTS
Cmtu	C4YZ	CMPM.L (A0)+,(A1)+	RE	RE	RTE
Ct vx	CX 2W	CMP.W A3,A2	RM rs,vx	RM T,LW1	MOVEM.L A6,LW1
Dcca p	DCSC L19	DBCS D2,L19	RMxt rs	RM4Z T-Y	" A1-A6,-(A0)
DSa v2	DSD WD1	DIVS WD1,D3	ROd v2	ROR WD1	ROR WD1
DUa v2	DUE 2MU	DIVU (A5),D4	ROd ma,sc	ROL 1D,C	ROL.B D2,D3
Ea	EF	SWAP D5	RR	RR	RTR
Egh	ESA	EXG A7,D0	RS	RS	RESET
FAt -i2	FAZ -23	LINK A0,-23	RXd v2	RXL WD1	ROXL WD1
FDt	FDT	UNLK A6	RXd ma,sc	RXR 4E,6	ROXR.L 6,D4
H i2	H $F0F0	STOP $F0F0	S v,i	S 1MW,32	SUB.B #32,(A3)
J v	J 1MYD-8	JMP -8(A1,D3)	Sa v	SA 2MX4G-8	SUB.W -8(A2,D6.L),D0
JS v	JS 1MX4G	JSR (A2,D6.W)	SBab	SBFE	SBCD D4,D5
LAt v	LAW 1MVI	LEA (A4)+,A3	SBtu	SBZY	SBCD -(A1),-(A0)
LSd v2	LSL WD1	LSL WD1	Scc v1	SGT BYT1	SGT BYT1
LSd ma,sc	LSR 1F,H	LSR.B D7,D5	SMa v	SME 4MWH	SUB.L D4,0(A3,D7)
M v,w	M LW1,J3	MOVE.L J3,LW1	St vx	SU 2F	SUB.W D5,A5
Ma v	MC 2,#$C	MOVE.B #$C,D2	SXmab	SX4HB	SUBX.L D1,D7
MCC v2	MCC $1A	MOVE $1A,CCR	SXmtu	SX1US	SUBX.B -(A7),-(A5)
MKa k	MKH -100	MOVEQ -100,D7	Sn v	S7 2MTI	SUBQ.W #7,(A6)+
MMa v	MMA WD2	MOVE.W D0,WD2	TE v	TE 1MV4X	TST.B 0(A4,A2.W)
MMt vx	MMV 4MU	MOVE.L A4,(A5)	TR vn	TR 4	TRAP 4
MR rs,vx	MR T,LW1	MOVEM.L LW1,A6	TS v1	TS 1G	TAS D6
MRxt rs	MR4Z A/H	" (A0)+,D0/D7	TV	TV	TRAPV
MSa v2	MSB 2MDW	MULS -(A3),D1	X v,i	X 1MW,$FF	EOR.B #$FF,(A3)
MSM v2	MSM 2MVI	MOVE SR,(A4)+	XCC i1	XCC $2	EOR.B #$2,SR
MSR v2	MSR 2MU	MOVE (A5),SR	XMa v	XMD LW1	EOR.L D3,LW1
MSt	MSX	MOVE A2,USP	XSR i2	XSR $400	EOR.W #$400,SR
Mt vx	MY 2MZ	MOVE.W (A0),A1	Xxa	X4C	EXT.L D2
MtS	MUS	MOVE USP,A5	Z v	Z 1MZ4Y+2	CLR.B 2(A0,A1.W)
MUa v2	MUC WD3	MULU WD3,D2			

Table IV

An example which shows the bug is that of a table of size greater than $82 (that is, hexadecimal 82), which is already sorted in descending order except that its last two bytes are $FF (and no other byte is $FF). In the given program, these final two bytes move up by one position in each pass through the table; and only two exchanges (the ones involving these two bytes) are made in each pass, since the table is otherwise already sorted in the proper order. Therefore, MB and A1 are executed only twice in each pass through the table. In the first pass, $FF is moved into B and this is then incremented by 1 *as a two-byte quantity*, producing $100. Subsequently, $FF is moved into the *rightmost byte* of B, leaving the rest of B undisturbed; this produces $1FF, which is then incremented to $200. Thus the final value of B is $200 at the end of the first pass, and, in general, it is 2 x $*k*00 at the end of the *k*th pass. At the end of the 128th pass, therefore, B will be zero; the TE then tests for zero; the BNE does *not* branch back; and the sort terminates with the two $FF bytes in positions 129 and 130 from the end, instead of in the first two positions, where they belong.

References

[1] Kane, G., D. Hawkins, and L. Leventhal, *68000 Assembly Language Programming*, Osborne/McGraw-Hill, Berkeley, 1981.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 524**

```
PGM_9_4A  MZ    LIST      (Move LIST, which is a 4-byte quan-
                           tity, to the Z-register)
          Z     2A        (Zero, or clear, two bytes of the
                           A-register)
          MA    1MZI      (Move, to the A-register, one byte of
                           memory, indexed by the Z-register,
                           and then increment the Z-register by 1)
          LAY   1MZA-1    (Load into the Y-register the address
                           in memory indexed by the Z register
                           and by the A register, minus 1)
SORT      Z     2B        (Zero, or clear, two bytes of the
                           B-register)
          MX    4Z        (Move, to the X-register, four bytes
                           of the Z-register)
NEXT      MA    1MXI      (Move, to the A-register, one byte in
                           memory, indexed by the X-register, and
                           then increment the X-register by 1)
          CA    1MX       (Compare the A-register with one byte
                           in memory, indexed by the X-register)
          BCC   NSWITCH   (Branch on carry clear to NSWITCH)
          MB    1MX       (Move, to the B-register, one byte in
                           memory indexed by the X-register)
          MMB   1MX-1     (Move to memory, from the B-register,
                           to one byte in memory, indexed by the
                           X-register minus 1)
          MMA   1MX       (Move to memory, from the A-register,
                           to one byte in memory, indexed by the
                           X-register)
          A1    2B        (Add 1 to two bytes of the B-register)
NSWITCH   CY    4X        (Compare the Y-register with four bytes
                           of the X-register)
          BHI   NEXT      (Branch on high, i.e., Y > X, to NEXT)
          TE    2B        (Test two bytes of the B register)
          BNE   SORT      (Branch to SORT on unequal to zero)
          R               (Return from subroutine)

                    REGISTERS

OLD   D0 D1 D2 D3 D4 D5 D6 D7 A0 A1 A2 A3 A4 A5 A6 A7

NEW   A  B  C  D  E  F  G  H  Z  Y  X  W  V  U  T  S
```

Table V

CP/M EXCHANGE

by Gene Head

Mail received after the first CP/M Exchange was encouraging. There is heavy interest in modem communication and program exchange. The most often asked question was, "How do I get started using my modem?" Well, it may be as simple as one, two three. I say *may* because each modem installation is different. If you have a basic understanding of the following three steps, you should be able to get signed-on to a Remote CP/M (RCP/M) System using the MBOOT3 program (listing begins on page 47).

1. Modem Status
2. Modem Input
3. Modem Output

If you understand how the CP/M functions CONSOLE STATUS, CONSOLE INPUT and CONSOLE OUTPUT operate, you also understand how the modem should operate. (Note: CONSOLE routines use only seven bits of data and the MODEM routines *must* use all eight bits.) Don't be put off if you fail to understand how to get the CP/M CONSOLE functions to operate. You'll need special assistance to get your modem functional, but I have a plan for you too!

MBOOT3 is a simple, stripped-down, receive-only version of Ward Christensen's MODEM program. The idea is to get MBOOT3 operating and use it to download (receive) a full MODEM program from an RCP/M system. If you have a modem, the software to simulate a terminal *and* the ability to "capture" received data into a file, just log on to an RCP/M and command the remote computer to list the MBOOT3 file using the CP/M command TYPE. Otherwise, spend an hour or so and type in MBOOT3.

Customize the hardware-dependent equates, assemble, and load MBOOT3. Follow the instructions on the facing page, remembering that each RCP/M system may be slightly different. Use your head. Don't be discouraged. Try to get the SYStem OPerator's help by using the CHAT command, or try ringing his bell by typing control-G. You can't hurt the remote system, and every RCP/M I've ever used has been very forgiving of errors and even untimely disconnections.

Modem Hint # 1

You will probably need cooperation from someone with a modem to test your system out before making long-distance calls to RCP/M's. If you can't get that special help, consider this: colleges with computer departments usually have a modem telephone for remote users. Without an account number and/or password you can't do much but you can at least verify that the basic communication, modem-to-modem, is operating. Execute the MBOOT3 program as directed in the source listing and then access any modem line by telephone. Try a few carriage returns to get the remote system "talking" to your system. Words like FULL and

HALF and HELP might get results. Check it out. . .

Special Assistance

Here is where the novice can be helped by the experienced modem user. If you have the MODEM program working on your system, send me a card with all the details about your specific hardware, the source listing of *just the hardware-dependent equates* and any unique hardware-dependent code, and how they are configured for your set-up.

Likewise, if you have the hardware but don't know exactly how to customize the software, send me a card with your specific hardware specifications. (Keep it simple!) I'll match your request with a known working system and send you the customization details. If I can't get an exact match, I'll send along the best data I have available. *Be sure to enclose an SASE with your request.* I'll do free photocopying (but no free postage) as long as this doesn't get too involved. . .

If you can offer help or need help, let the "CP/M Exchange" know! Be sure to include your disk format and complete modem configuration. Send your complete name, address, city, state, zip, and phone with area code.

Exchange Mail

The first response to the CP/M Exchange came from *DDJ* contributor Bob Blum. Bob suggested electronic mail for communication. If the interest is there, we might be able to compile a directory of those using this form of communication. I will make my RCP/M system available if there is serious interest. Anyone else have ideas along these lines? Bob is also preparing a contribution on keyboard buffering in a modified CCP for CP/M that will be included in a future "CP/M Exchange."

The most specific letter seeking help came from Charles Henderson of Midland, Texas. Charles is interested in why SPEED and FAST won't work with CP/M 2.x. He suspects the track buffering is involved but doesn't understand all the implications. A definitive article on track and sector skewing, interleaving, sector tables, etc. would be most appropriate. If you have a solid understanding of these principles, write a paper and send it in! Share with others what you have learned.

Dana Trout of Goleta, California, notes that CP/M Function #37 does not work as described in the manual. This function to reset the disk drives will reset all drives *except* the current default drive. If the current default drive is set to R/O and function thirty-seven is executed, the default drive is still in R/O! Anyone have a fix and/or explanation for this?

Roland Lupient, KB9RR, has ham radio gear, an Altair 8800 and CP/M User's Group volume 41. He would like to hear from anyone who can help interface the ham gear and computer (address: 1953 Graham Lane, Mosinee, WI 54455).

Finally, William Burnett of Sinton, Texas, wonders if there are any Superbrain users out there who have access to the CP/M User's Library? The same question can be asked about North Star, Osborne, and all the other five-inch formats. If you can copy the CP/M User's Library to any of the many five-inch disk formats, let me know. I suspect you will be in great demand, and while you may not make a lot of money at this, a couple of bucks copying fee seems reasonable.

That covers it for now. Every letter to the "CP/M Exchange" has received either a personal response or was noted in this column. If you have something to contribute, a question or comment, let's hear from you!

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 528**

Downloading MODEM Using MBOOT3

First, execute MBOOT3 as instructed in the source listing (begins on page 47). It will sign on, but there will be no screen activity after the sign-on messages. In the TERMINAL mode, all displayed console characters come from the remote system. Even keys from your keyboard are "echoed" back, so if there is no remote system to echo back the keystrokes, you won't see them on your CRT.

Now, call a remote CP/M System (see list on page 44), when you hear the remote modem carrier tone, connect your modem to the phone line. The two modems are now connected and your computer is "talking" to the remote computer. Press the Return key once a second until you see the remote computer's sign-on message, usually asking how many nulls you need. (Answer 0 to the nulls question if you're using a video terminal.)

Follow the host computer's instructions until you finally reach the CP/M operating level with the expected A>. You are now in control of the remote machine.

To find the MODEM program type:

```
DIR MODEM *.*
```

This will show you the exact name of the MODEM files available. You should see in the listing the MODEM.ASM file or some similar source code file.

To receive the file type:

```
XMODEM filename.type
```

The remote system should tell you that XMODEM is ready to send the requested file, then it will wait. There will be no further screen activity from the host computer. It will look as though you have lost control, but don't fret. The host is waiting for your MBOOT3 program to send special protocol characters to begin the transfer and won't respond to your keyboard characters until the transfer is completed or aborted.

Now that you know the host is ready to send and is waiting for you to be ready to receive, press your ESCape key. This key tells MBOOT3 to receive the file the host is sending.

Most CP/M commands are available to remote users, so use the remote system just as you would your local computer. Be sure to sign off by typing BYE and disconnect your modem from your phone line, if it isn't automatically disconnected by your software.

As with any new operation, practice and experience are the best teachers.

CP/M Exchange *(Continued from page 43)*

Remote CP/M Software Exchange Systems
List # 24
Last Revised March 21, 1982

List #24 revised and updated courtesy of Hyde Park RCPM, CP/M-Net, and Mississauga RCPM.

This is part of a CP/M file called RCPMLIST.xx. It is found on most RCP/M systems. The systems presented here are representative of what you can expect to find. Each was "up" as of February, 1982, unless otherwise noted. Log on to any system and then get a copy of this complete file for an accurate directory.

This is a partial list.

California

(Best)
Bakersfield, CA
CP/M-Net[tm], (805) 527-9321, Kelly Smith
No A.L.D.S.; 1900-2300 (PST) Monday-Friday, 1900 Friday to 0700 Monday
110-600+ baud
20 Mb of files on 2 hard disks (=8 logical disks)
System now includes SIG/M Vol. 1-10 = E:, SIG/M Vol. 11-20 = F:, SIG/M Vol 21-25 = G:. XMODEM 'DISKMENU.DOC' for entire system directory (over 2100 files now available!).

Garden Grove, CA
G.F.R.N. Data Exchange RBBS, (714) 534-1547, Doug Laing
24 hour operation
300 and 1200 baud
5 Mb of files on 4 drives
Special interest in amateur radio and Apple/CPM. This is the second G.F.R.N. system.

Larkspur, CA
Larkspur RBBS/RCPM (415) 461-7726, Jim C.
24 hour operation
Up as of 3/20/81
110, 300, 450, 710 baud; SPRINT, ITT, MCI
2+ Mb on 2 drives
The system carries general and new CP/M software. System now running MP/M and plans are afoot to install a second telephone line, making it the first multi-user remote system. SYSOP will assist others in bringing up MP/M remotely.

Palos Verdes, CA
G.F.R.N. Data Exchange RBBS, (213) 541-2503, Skip Hansen
24 hour operation
300 and 1200 baud; SPRINT, MCI, ITT
2.4 Mb of files on 2 drives
Standard CP/M software with special interest in ham radio-related programs. Soon (with MP/M) will also be reachable through 450 MHz radio.

San Diego, CA
San Diego RCPM, (714) 271-5615, Brian Kantor
No call back; 24 hour operation
300 and 1200 baud; ITT, SPRINT, MCI
2.4 Mb of files on 2 drives
Now with 212-compatible modem; yet another 1200-baud.

East Central

Allentown, PA
Allentown RBBS/RCPM System, (215) 398-3937, Bill Ernest
No call back; 24 hour operation
110, 300, 450, 600, 710 baud; SPRINT and ITT
4.25 Mb of files on hard disk (=4 logical disks)
General CP/M software. Bulletin board of the Lehigh Valley Computer Club.

Grafton, VA
Grafton, VA RBBS, (804) 898-7493, Dave Holmes
No call back, no A.L.D.S; 24 hour operation
300 baud
200 Kb of files on 2 drives
Carries CP/M, TRS-80 and Apple software; plans for setting up a dual system (on one line) with an LNW-80 as well as the CP/M computer. Active as bulletin board.

McClean, VA
RLP RCP/M, McClean, VA, (703) 524-2549, Bob Plouffe
No call back; 24 hour operation
SPRINT and MCI
4 (N*) drives with 640 Kb of files
Running CBBS for messages. New system.

Midwest

Chicago (area), IL
Calamity Cliffs Computer Center, (312) 234-9257
No call back; 1400-0200 daily
300, 450, 600 baud; ITT, SPRINT, MCI
11 Mb of files on a hard disk and 2 floppies
Many of the CPMUG and SIG/M programs available by request.

Chicago, IL
HUG-CBBS, (312) 671-4992, Paul Mayer, Dave Leonard
No call back; 2300 to 1900, 7 days/week
300 baud; SPRINT, ITT, MCI
2 Mb of files on 2 drives
H89-based, operated for the Heath-Zenith Users' Group and with a special interest in H19- and H89-adapted (as well as general CP/M) software.

Hyde Park, IL
Hyde Park RCPM/RBBS, (312) 955-4493, Ben Bronson
No call back; 0100-1700 daily
110, 300, 450, 600, 710 baud; SPRINT, ITT, MCI
2 Mb of files on 2 drives
Special interest in hardware and software reviews, C programs, and very recent releases of standard programs. SYSOP now testing substantial upgrade of RBBS programs.

Royal Oak, MI
Royal Oak CP/M, (313) 759-6569, Keith Petersen

Call back; 24 hour operation
110, 300, 450, 600 baud – 1200 baud modem now available on request; use CHAT or leave a message if you want the 212A switched in – ITT, SPRINT, MCI
600 Kb on two floppy drives and 10 Mb on hard disk (=2 logical drives)
Emphasis on new programs and recent updates of standard programs.

Westland, MI
Westland, MI RBBS/RCPM, (313) 729-1905, Ron Fowler
Call back; 24 hour operation
110, 300, 450 and 600 baud; SPRINT, MCI, ITT
1.4 Mb of files on 2 drives
Emphasis on very recent releases.

Northeast

Lexington, MA
Superbrain RCPM, (617) 862-0781, Paul Kelly
1900-0700 Weekdays, 24 hours weekends
110, 300, 1200 baud; SPRINT, ITT, MCI
300 Kb files on-line
Special interest in Superbrain-adapted CP/M programs.

Bearsville, NY
Bearsville Town SJBBS, (914) 679-6559, Hank Szyszka
No call back, no A.L.D.S.
110, 300, 450, 600, 710 baud
2 Mb of files on 4 drives
Installing MP/M. All CP/MUG programs available by request. General CP/M software.

Rochester, NY
Rochester RBBS, (716) 223-1100, Arnie McGall
No call back; 24 hour operation
110 and 300 baud; SPRINT, MCI, ITT
1.8 Mb of files on 3 drives
S-100 based. General CP/M software. The standard RBBS/RCPM system co-exists with a separate passworded message system called DataStar, which can be entered from CP/M but runs on a separate computer. 600 baud capability expected soon.

Hamilton, ON
Hamilton Area Packet Radio Network (HAPN), (416) 335-6620, Stu Beal
No A.L.D.S; 24 hour operation
110, 300, 450, 600, 710 baud
@? Kb files on-line
New system. System is also linked to radio network and may be accessed via ham radio. Special interest in radio software.

Toronto, ON
Mississauga, Ontario RCPM, (416) 826-5394, Jud Newell
No A.L.D.S.; 24 hour operation
110, 300, 450, 600, 710, 1200 baud
20 Mb hard disk now on-line 24 hours a day
1200 baud Vadic/Bell 212A standard both supported. 300/1200 baud modem available Monday-Friday, PMMI weekends. XMODEM, DISKMENU.DOC and MAST.CAT for details of over 3000 available files.

(Continued on page 46)

CP/M Exchange *(Continued from page 45)*

System now restricted to CP/M users only by use of a familiarity question.

Toronto, ON
Mississauga, Ontario HUG-RCP/M, (416) 273-3011
No A.L.D.S.; 1800-0600 weekdays, 24 hours weekends
110, 300, 450, 600, 710 baud
2+ Mb of files on 5 drives
Toronto Heath Users' Group.

South

Huntsville, AL
NACS/UAH RBBS/RCPM, (205) 895-6749, Don Wilkes
Call back, no A.L.D.S.; 24 hour operation
110, 300, 450, 600 baud
700 Kb files on 4 drives
Run for North Alabama Computer Society at University of Alabama; general CP/M software.

Louisville, KY
Louisville RBBS/RCPM, (502) 245-7811, Mike Jung
No call back; 0900-2100 weekdays, 24 hours weekends
300 baud; SPRINT, MCI
2.5 Mb of files on 5 drives
Heath/Zenith-based. Emphasis on BASIC software. Some HDOS stuff available for downloading.

Fort Mill, SC
Fort Mill RIBBS, (803) 547-6576, Bill Taylor
Up as of 08/15/81
No call back, no A.L.D.S.; 24 hour operation
300 and 1200 baud
3 Mb of files on 2 drives
Heath/Zenith-based with 212-compatible modem. The system carries ham stuff, general software, and on-line games. The fourth 1200-baud RCM.

DDJ

(Listing begins at right)

MBOOT.ASM

(Text begins on page 42)

```
;          Received from Kelly Smith's CP/M-NET (tm)
;          For publication in Doctor Dobb's Journal
;
;          ASSEMBLED, LOADED AND EXECUTED AS SHOWN
;               (except for custom equates)
;
; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;
;
; MBOOT.ASM ver 3.0
; (revised 5 Nov 80)
;
;MODEM BOOT PROGRAM by Keith Petersen, W8SDZ.
;Thanks to John Taylor for idea of incorporating
;simple terminal routine.  Based on MODEM.ASM V2.0,
;by Ward Christensen with enhancements from MODEM527.
;
;CP/M - CP/M file transfer program (receive-only)
;
;This program is intended for use as a way to
;initially transfer MODEM.COM or MODEM.ASM to
;a new user.  It may be used with almost any
;modem (see equates).  If PMMI or D.C. Hayes modem
;is used, then uses 'ORIGINATE' mode, 300 baud.
;
;A simple terminal routine at start of program allows
;user to communicate with a remote system prior to
;receiving a file to make it possible to down-load
;a file without intervention on the part of the host
;system's operator.
;
;COMMANDS:  MBOOT FILENAME.FILETYPE
; or   MBOOT A:FILENAME.FILETYPE
; or   MBOOT B:FILENAME.FILETYPE
;
;The program will operate as a dumb terminal until
;an 'ESC' key is typed (ctrl-[).  It then branches
;to the receive routine.  The user may also exit
;to CP/M without opening the receive file by typing
;ctrl-E from the terminal.  The values for the escape
;and exit keys may be changed in accordance with the
;needs of the user - some keyboards do not have the
;'ESC' key and/or provision for control characters.
;See equates.
;
;NOTE: Comments for the source code and tabs have been
;removed to make this file easier to transport from one
;system to another.  (KBP)
;
;NOTE:   NOTE:     TABS have been re-expanded for DDJ format
;
FALSE    EQU       0
TRUE     EQU       NOT FALSE
;
STDCPM   EQU       TRUE     ;TRUE, IS STANDARD CP/M
ALTCPM   EQU       FALSE    ;TRUE, IS ALTERNATE CP/M FOR H8 OR TRS80
;
         IF        STDCPM
BASE     EQU       0
         ENDIF
;
```

(Continued on column 2)

```
         IF        ALTCPM
BASE     EQU       4200H
         ENDIF
;
EXITCHR  EQU       05H       ;CTL-E TO EXIT TERM MODE TO CP/M
ESC      EQU       1BH       ;ESCAPE TO EXIT TERM MODE TO FILE RCVE
;
FASTCLK  EQU       FALSE     ;TRUE FOR 4 MHZ CLOCK
;
PMMI     EQU       TRUE      ;TRUE, IS PMMI MODEM
DCH      EQU       FALSE     ;TRUE, IS D.C. HAYES MODEM
;
;IF YOU ARE USING AN EXTERNAL MODEM (NOT S-100 PLUG-IN)
;CHANGE THESE EQUATES FOR YOUR MODEM PORT REQUIREMENTS
;
INITREQ  EQU       FALSE     ;TRUE, IF MODEM PORT INIT. REQ'D
INITC1   EQU       0AAH      ;1ST INIT CHAR TO CTL PORT FOR USART
INITC2   EQU       40H       ;2ND  "
INITC3   EQU       4EH       ;3RD  "
INITC4   EQU       37H       ;4TH  "
;
         IF NOT PMMI AND NOT DCH
MODCTLP  EQU       02H       ;PUT YOUR MODEM CONTROL PORT HERE
MODSNDB  EQU       80H       ;YOUR BIT TO TEST FOR SEND
MODSNDR  EQU       80H       ;YOUR VALUE WHEN READY TO SEND
MODRCVB  EQU       40H       ;YOUR BIT TO TEST FOR RECEIVE
MODRCVR  EQU       40H       ;YOUR VALUE WHEN READY TO RECEIVE
MODDATP  EQU       03H       ;YOUR MODEM DATA PORT
         ENDIF               ;END OF EXTERNAL MODEM EQUATES
;
         IF PMMI
MODCTLP  EQU       0C0H
MODSNDB  EQU       1
MODSNDR  EQU       1
MODRCVB  EQU       2
MODRCVR  EQU       2
MODDATP  EQU       0C1H
BAUDRP   EQU       0C2H
MODCTL2  EQU       0C3H
ORIGMOD  EQU       1DH
ANSWMOD  EQU       1EH
         ENDIF
;
         IF DCH
MODCTLP  EQU       82H
MODSNDB  EQU       2
MODSNDR  EQU       2
MODRCVB  EQU       1
MODRCVR  EQU       1
MODDATP  EQU       80H
MODCTL2  EQU       81H
ORIGMOD  EQU       87H
ANSWMOD  EQU       83H
         ENDIF
;
ERRLIM   EQU       10
;
SOH      EQU       1
EOT      EQU       4
ACK      EQU       6
NAK      EQU       15H
CAN      EQU       18H
LF       EQU       10
CR       EQU       13
;
BDOS     EQU       BASE+5
```

(Continued on next page)

MBOOT.ASM

(Listing continued, text begins on page 42)

```
FCB      EQU     BASE+5CH
;
         ORG     BASE+100H
;
         LXI     H,0
         DAD     SP
         SHLD    STACK
         LXI     SP,STACK
         CALL    INITADR
         CALL    ILPRT
         DB      'MBOOT as of '
         DB      '11/5/80',CR,LF,0
         LDA     FCB+1
         CPI     ' '
         JNZ     TERMI
         CALL    ILPRT
         DB      '++NO FILE NAME SPECIFIED++',CR,LF,0
         JMP     EXIT
;
TERMI    CALL    INITMOD
         IN      MODDATP
         IN      MODDATP
         CALL    ILPRT
         DB      CR,LF,'TERMINAL MODE',CR,LF
         DB      'CTL-E exits to CP/M, ESC starts file transfer'
         DB      CR,LF,0
;
TERM     CALL    STAT
         JZ      TERML
         CALL    KEYIN
         CPI     EXITCHR
         JZ      EXIT
         CPI     ESC
         JZ      RCVFIL
         OUT     MODDATP
;
         IF NOT DCH
TERML    IN      MODCTLP
         ENDIF
;
         IF DCH
TERML    IN      MODCTL2
         ENDIF
;
         ANI     MODRCVB
         CPI     MODRCVR
         JNZ     TERM
         IN      MODDATP
         ANI     7FH
         CALL    TYPE
         JMP     TERM
;
RCVFIL   CALL    ERASFIL
         CALL    MAKEFIL
         CALL    ILPRT
         DB      'FILE OPEN, READY TO RECEIVE',CR,LF,0
;
RCVLP    CALL    RCVSECT
         JC      RCVEOT
         CALL    WRSECT
```

(Continued on top of page 48)

```
         CALL    INCRSNO
         CALL    SENDACK
         JMP     RCVLP
;
RCVEOT   CALL    WRBLOCK
         CALL    SENDACK
         CALL    CLOSFIL
         CALL    ERXIT
         DB      CR,LF,'TRANSFER COMPLETE$'
;
RCVSECT  XRA     A
         STA     ERRCT
;
RCVRPT   MVI     B,10
         CALL    RECV
         JC      RCVSERR
         CPI     SOH
         JZ      RCVSOH
         ORA     A
         JZ      RCVRPT
         CPI     EOT
         STC
         RZ
;
RCVSERR  MVI     B,1
         CALL    RECV
         JNC     RCVSERR
         MVI     A,NAK
         CALL    SEND
         LDA     ERRCT
         INR     A
         STA     ERRCT
         CPI     ERRLIM
         JC      RCVRPT
;
RCVSABT  CALL    CLOSFIL
         CALL    ERXIT
         DB      '++UNABLE TO RECEIVE BLOCK'
         DB      CR,LF,'++ABORTING++$'
;
RCVSOH   MVI     B,1
         CALL    RECV
         JC      RCVSERR
         MOV     D,A
         MVI     B,1
         CALL    RECV
         JC      RCVSERR
         CMA
         CMP     D
         JZ      RCVDATA
         JMP     RCVSERR
;
RCVDATA  MOV     A,D
         STA     RCVSNO

         MVI     C,0
         LXI     H,BASE+80H
;
RCVCHR   MVI     B,1
         CALL    RECV
         JC      RCVSERR
         MOV     M,A
         INR     L
         JNZ     RCVCHR
         MOV     D,C
         MVI     B,1
         CALL    RECV
```

(Continued on page 49, column 1)

```
        JC      RCVSERR
        CMP     D
        JNZ     RCVSERR
        LDA     RCVSNO
        MOV     B,A
        LDA     SECTNO
        CMP     B
        JZ      RECVACK
        INR     A
        CMP     B
        JNZ     ABORT
        RET
;
RECVACK CALL    SENDACK
        JMP     RCVSECT
;
SENDACK MVI     A,ACK
;
SEND    PUSH    PSW
        ADD     C
        MOV     C,A
;
        IF NOT DCH
SENDW   IN      MODCTLP
        ENDIF
;
        IF DCH
SENDW   IN      MODCTL2
        ENDIF
;
        ANI     MODSNDB
        CPI     MODSNDR
        JNZ     SENDW
        POP     PSW
        OUT     MODDATP
        RET
;
ABORT   LXI     SP,STACK
;
ABORTL  MVI     B,1
        CALL    RECV
        JNC     ABORTL
        MVI     A,CAN
        CALL    SEND
;
ABORTW  MVI     B,1
        CALL    RECV
        JNC     ABORTW

        MVI     A,'      '
        CALL    SEND
        CALL    ERXIT
        DB      'MBOOT PROGRAM CANCELLED$'
;
INCRSNO LDA     SECTNO
        INR     A
        STA     SECTNO
        RET
;
ERASFIL LXI     D,FCB
        MVI     C,17
        CALL    BDOS
        INR     A
        RZ
        CALL    ILPRT
        DB      '++FILE EXISTS, TYPE Y TO ERASE:',0
```

(Continued on column 2)

```
        CALL    KEYIN
        PUSH    PSW
        CALL    TYPE
        CALL    CRLF
        POP     PSW
        ANI     5FH
        CPI     'Y'
        JNZ     MXIT
        LXI     D,FCB
        MVI     C,19
        JMP     BDOS
;
MAKEFIL LXI     D,FCB
        MVI     C,22
        CALL    BDOS
        INR     A
        RNZ
        CALL    ERXIT
        DB      '++ERROR - CAN''T MAKE FILE',CR,LF
        DB      '++DIRECTORY MUST BE FULL$'
;
CLOSFIL LXI     D,FCB
        MVI     C,16
        CALL    BDOS
        INR     A
        RNZ
        CALL    ERXIT
        DB      '++CAN''T CLOSE FILE$'
;
WRSECT  LHLD    SECPTR
        XCHG
        LXI     H,BASE+80H
        CALL    MOVE128
        XCHG
        SHLD    SECPTR
        LDA     SECINBF
        INR     A
2STA    SECINBF
        CPI     16
        RNZ
;
WRBLOCK LDA     SECINBF
        ORA     A
        RZ
        MOV     C,A
        LXI     D,DBUF
;
DKWRLP  PUSH    H
        PUSH    D
        PUSH    B
        MVI     C,26
        CALL    BDOS
        LXI     D,FCB
        MVI     C,21
        CALL    BDOS
        POP     B
        POP     D
        POP     H
        ORA     A
        JNZ     WRERR
        LXI     H,80H
        DAD     D
        XCHG
        DCR     C
        JNZ     DKWRLP
        XRA     A
        STA     SECINBF
```

(Continued on column 3)

```
        LXI     H,DBUF
        SHLD    SECPTR
;
RSDMA   LXI     D,BASE+80H
        MVI     C,26
        JMP     BDOS
;
WRERR   CALL    RSDMA
        CALL    ILPRT
        DB      '++ERROR WRITING FILE',CR,LF,0
        JMP     ABORT
;
RECV    PUSH    D
;
        IF      FASTCLK
        MOV     A,B
        ADD     A
        MOV     B,A
        ENDIF
;
MSEC    LXI     D,50000
;
        IF NOT DCH
MWTI    IN      MODCTLP
        ENDIF
;
        IF      DCH
MWTI    IN      MODCTL2
        ENDIF
;
        ANI     MODRCVB
        CPI     MODRCVR
        JZ      MCHAR
        DCR     E
        JNZ     MWTI
        DCR     D
        JNZ     MWTI
        DCR     B
        JNZ     MSEC
        POP     D
        STC
        RET
;
MCHAR   IN      MODDATP
        POP     D
        PUSH    PSW
        ADD     C
        MOV     C,A
        POP     PSW
        ORA     A
        RET
;
INITADR LHLD    BASE+1
        LXI     D,3
        DAD     D
        SHLD    VSTAT+1
        DAD     D
        SHLD    VKEYIN+1
        DAD     D
        SHLD    VTYPE+1
        RET
;
INITMOD EQU     $
;
        IF      INITREQ
```

(Continued on next page)

MBOOT.ASM

(Listing continued, text begins on page 42)

```
         MVI     A,INITC1
         OUT     MODCTLP
         NOP ! NOP ! NOP
         NOP ! NOP
         MVI     A,INITC2
         OUT     MODCTLP
         NOP ! NOP ! NOP
         NOP ! NOP
         MVI     A,INITC3
         OUT     MODCTLP
         NOP ! NOP ! NOP
         NOP ! NOP
         MVI     A,INITC4
         OUT     MODCTLP
         NOP ! NOP ! NOP
         NOP ! NOP
         ENDIF
;
         IF      PMMI
         MVI     A,52
         OUT     BAUDRP
         MVI     A,7FH
         OUT     MODCTL2
         ENDIF
;
         IF      DCH
         MVI     A,16H
         OUT     MODCTL2
         ENDIF
;
         IF      PMMI OR DCH
;
         LXI     H,4000
;
OFFDLY   DCR     L
         JNZ     OFFDLY
         DCR     H
         JNZ     OFFDLY
         MVI     A,ORIGMOD
         OUT     MODCTLP
         ENDIF
;
         RET                     ;<--THIS RET MUST BE HERE
;
CRLF     MVI     A,CR
         CALL    TYPE
         MVI     A,LF
;
TYPE     PUSH    PSW
         PUSH    B
         PUSH    D
         PUSH    H
         MOV     C,A
VTYPE    CALL    $-$
         POP     H
         POP     D
         POP     B
         POP     PSW
         RET
;
```

(Continued on column 3)

```
STAT     PUSH    B
         PUSH    D
         PUSH    H
VSTAT    CALL    $-$
         POP     H
         POP     D
         POP     B
         ORA     A
         RET
;
KEYIN    PUSH    B
         PUSH    D
         PUSH    H
VKEYIN   CALL    $-$
         POP     H
         POP     D
         POP     B
         ANI     7FH
         RET
;
ILPRT    XTHL
;
ILPLP    MOV     A,M
         ORA     A
         JZ      ILPRET
         CALL    TYPE
         INX     H
         JMP     ILPLP
;
ILPRET   XTHL
         RET
;
ERXIT    POP     D
         MVI     C,9
         CALL    BDOS
;
MXIT     CALL    ILPRT
         DB      CR,LF,'DON''T FORGET TO DISCONNECT MODEM'
         DB      CR,LF,0
;
EXIT     LHLD    STACK
         SPHL
         RET
;
MOVE128  MVI     B,128
;
MOVE     MOV     A,M
         STAX    D
         INX     H
         INX     D
         DCR     B
         JNZ     MOVE
         RET
;
RCVSNO   DB      0
SECTNO   DB      0
ERRCT    DB      0
EOFLG    DB      0
SECPTR   DW      DBUF
SECINBF  DB      0
         DS      60
STACK    DS      2
DBUF     EQU     $
;
         END
```

End Listing

16-BIT SOFTWARE TOOLBOX

by Ray Duncan

Late News for 8087 Fanciers

Intel recently dropped the price of the Intel 8087 numeric processor to $230 in single-unit quantities. Some high-level software tools to exploit the 8087 are also beginning to appear. In addition to my own company's version of Forth which can use the 8087 co-processor, Lifeboat Associates is marketing the Lattice C Compiler which includes an 8087 runtime library, and rumor tells of a UCSD Pascal support file named 8087.PLS which is not yet officially announced.

16-Bit Tools, Installment # 2

The accompanying listing is an 8086/88 assembly language subroutine that returns the sine or cosine of an angle to four significant digits. The precision is adequate for most graphics applications, and the routines are implemented by a combination of deduction and table-lookup techniques that are extremely fast. The original version of these routines was coded in Forth by John James and placed in the public domain; interested readers are referred to his article in *Forth Dimensions*, volume 4, no. 1.

Readers who also need a tangent function can derive it easily:

$$\tan(x) = \frac{\sin(x) * 10000}{\cos(x)}$$

But watch out: finding tangents in this way for angles greater than seventy-two degrees will lead to a divide overflow, which causes a hardware interrupt on the 8086/88.

A more precise (but slower) method of finding sine, cosine and tangent through a simple polynomial expansion will be presented in a later column.

Product Report: Microsoft RAMCard with RAMDrive

This integrated combination of hardware and software can be installed in any IBM Personal Computer, and will markedly enhance its operation. "RAMDrive" refers to a technique of mapping a disk directory and file structure onto a portion of the machine's random access memory. The operating system drivers are modified so that the simulated disk can be accessed by application programs like any other peripheral device. Since there are no moving parts, data can be retrieved nearly instantaneously. As many as three RAMCards may be installed, yielding a maximum of 768 Kbytes of RAMDrive storage.

The package contains a memory expansion board of average-quality construction socketed for 256 Kbytes with parity, a diskette of utility programs and a sixty-page manual. The product is available in several versions and prices, depending on the amount of RAM initially supplied. The user can upgrade a partially filled card by simply plugging in more dynamic memory chips.

Installing the memory card in the IBM PC is quite simple, and can be done in a few minutes by anyone who can read and handle a screwdriver. The manual is clear and complete, and has plenty of helpful troubleshooting tips.

Next, you use a memory test provided on the Microsoft diskette to make sure

the RAMCard is working properly. This program is a classic of its kind. It was obviously written by a programmer who was told to make the product "user-friendly," and who thought he could meet that objective with liberal use of reverse-video, blinking text, complicated screen-oriented presentations and, of course, using the special-function keys at any cost. For example, the user is asked to respond to one of the program's setup questions by pushing either the F9 function button or the space bar, neither of which are conveniently located or have the slightest mnemonic value. In spite of its veneer of razzle-dazzle, the program suffers from major design faults in the human interface that make it awkward to use, such as a lack of backspace capability should you be unfortunate enough to make an error in numeric entry. The program and documentation don't provide the smallest morsel of information about the testing process itself, and the only way to terminate the test once it is running is to reset the computer. This is not my idea of a classy program.

After you are convinced that the memory card is working correctly, you run the configuration program, which patches the PC-DOS operating system and makes the extra memory available as a simulated disk device. This is a simple procedure and requires only a few seconds – afterwards, you can copy the revised operating system to your various working disks. You might wonder what would prevent someone from pirating the configuration programs and using them with other companies' less-expensive memory cards. Never fear, Microsoft has provided against this eventuality: part of the necessary software is in a programmed read-only memory chip installed right on the RAMCard.

Using the RAMDrive capability can result in astonishing improvements in performance for programs which are disk I/O bound. For example, a simple Forth program to sequentially read through a 100 Kbyte file, accessing it as 1024-byte records, completed in 24.9 seconds using a regular 5¼-inch floppy disk drive but only required one second on the RAMDrive.

The RAMCard and RAMDrive package costs $495 with 64Kbytes installed, and each additional 64Kbytes costs you $200 from Microsoft up to a grand total of $1095 for a complete 256Kbyte card. Here is where a little careful shopping can save you a bundle of money. I bought the minimum 64Kbyte card from Microsoft via Computerland, and obtained the twenty-seven dynamic RAM (type 4164) chips necessary to fill up the card for $227 from BG Micro in Garland, Texas. Thus, I saved $373 as compared to buying the RAM expansions from Microsoft.

Note that if you do not want the RAMDrive feature of this product, there are many memory cards available from reputable companies at considerably lower cost. But if you really need increased speed from your IBM PC, but aren't yet ready to take the plunge on a hard disk, then this product is great!

(Listing begins at right)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 530**

8086 Trig

(Text begins on page 51)

```
                                  title   'Trig Lookup Functions'
                                  pagewidth       79
                          ;
                          ; Trig lookup functions for 8086/88
                          ; adapted by Ray Duncan from a public
                          ; domain FORTH routine written by John James.
                          ;
                          ;
                          ; 'Trig' is a common routine used by 'sin'
                          ; and 'cos' to reduce an angle into the
                          ; range 0-90 degrees then extract the
                          ; appropriate function value from the table.
                          ;
0000 8BD8                 trig:   mov     bx,ax      ;bx <- degrees
0002 83FB5A                       cmp     bx,90      ;if > 90 degrees
0005 7E06           000D          jle     trig1      ;reduce the angle
0007 81EBB400                     sub     bx,180
000B F7DB                         neg     bx
000D D1E3                 trig1:  sal     bx,1       ;table index = 2*deg.
                                                     ;now extract value
000F 2E8B873F00                   mov     ax,sintbl [bx]
0014 C3                           ret                ;back to caller
                          ;
                          ;  Cosine lookup:  add 90 degrees to
                          ;  argument and use sine lookup.
                          ;  call with:       ax = degrees
                          ;  returns:         ax = cosine * 10000
                          ;  other registers preserved.
                          ;
0015 055A00               cos:    add     ax,90
                          ;
                          ;  Sine lookup:  reduce to angle in
                          ;  range 0-359 degrees, then call
                          ;  'trig' to extract function value
                          ;  from table.
                          ;  call with:      ax = degrees
                          ;  returns:        ax = sine * 10000
                          ;  other registers preserved.
                          ;
0018 52                   sin:    push    dx         ;save registers
0019 53                           push    bx
001A 99                           cwd                ;deg -> double prec.
001B BB6801                       mov     bx,360     ;reduce angle to
001E F7FB                         idiv    bx         ;range 0-359 degrees
0020 8BC2                         mov     ax,dx      ;let ax=remainder
0022 0BC0                         or      ax,ax      ;is angle negative?
0024 7903           0029          jns     sin2       ;no, jump
0026 056801                       add     ax,360     ;yes,make it positive
0029 3DB400               sin2:   cmp     ax,180     ;now reduce angle to
002C 7E0B           0039          jle     sin3       ;range 0-180 degrees
002E 2DB400                       sub     ax,180     ;angle was > 180 deg.
0031 E8CCFF         0000          call    trig       ;look up function value

0034 F7D8                         neg     ax
0036 E90300         003C          jmp     sin4
0039 E8C4FF         0000  sin3:   call    trig       ;angle was <= 180 deg
003C 5B                   sin4:   pop     bx         ;restore registers
003D 5A                           pop     dx
003E C3                           ret                ;back to caller
                          ;
                          ; lookup table for trig functions
                          ;
003F 0000                 sintbl  dw      0          ;0 degrees
0041 AF00                         dw      175        ;1
0043 5D01                         dw      349        ;2
0045 0B02                         dw      523        ;3
0047 BA02                         dw      698        ;4
0049 6803                         dw      872        ;5
004B 1504                         dw      1045       ;6
004D C304                         dw      1219       ;7
004F 7005                         dw      1392       ;8
0051 1C06                         dw      1564       ;9
0053 C806                         dw      1736       ;10
0055 7407                         dw      1908       ;11
0057 1F08                         dw      2079       ;12
0059 CA08                         dw      2250       ;13
005B 7309                         dw      2419       ;14
005D 1C0A                         dw      2588       ;15
005F C40A                         dw      2756       ;16
0061 6C0B                         dw      2924       ;17
0063 120C                         dw      3090       ;18
0065 B80C                         dw      3256       ;19
0067 5C0D                         dw      3420       ;20
```

(Continued on next page)

8086 Trig

(Listing continued, text begins on page 51)

```
0069 000E        dw      3584    ;21
006B A20E        dw      3746    ;22
006D 430F        dw      3907    ;23
006F E30F        dw      4067    ;24
0071 8210        dw      4226    ;25
0073 2011        dw      4384    ;26
0075 BC11        dw      4540    ;27
0077 5712        dw      4695    ;28
0079 F012        dw      4848    ;29
007B 8813        dw      5000    ;30
007D 1E14        dw      5150    ;31
007F B314        dw      5299    ;32
0081 4615        dw      5446    ;33
0083 D815        dw      5592    ;34
0085 6816        dw      5736    ;35
0087 F616        dw      5878    ;36
0089 8217        dw      6018    ;37
008B 0D18        dw      6157    ;38
008D 9518        dw      6293    ;39
008F 1C19        dw      6428    ;40
0091 A119        dw      6561    ;41
0093 231A        dw      6691    ;42

0095 A41A        dw      6820    ;43
0097 231B        dw      6947    ;44
0099 9F1B        dw      7071    ;45
009B 191C        dw      7193    ;46
009D 921C        dw      7314    ;47
009F 071D        dw      7431    ;48
00A1 7B1D        dw      7547    ;49
00A3 EC1D        dw      7660    ;50
00A5 5B1E        dw      7771    ;51
00A7 C81E        dw      7880    ;52
00A9 321F        dw      7986    ;53
00AB 9A1F        dw      8090    ;54
00AD 0020        dw      8192    ;55
00AF 6220        dw      8290    ;56
00B1 C320        dw      8387    ;57
00B3 2021        dw      8480    ;58
00B5 7C21        dw      8572    ;59
00B7 D421        dw      8660    ;60
00B9 2A22        dw      8746    ;61
00BB 7D22        dw      8829    ;62
00BD CE22        dw      8910    ;63
00BF 1C23        dw      8988    ;64
00C1 6723        dw      9063    ;65
00C3 AF23        dw      9135    ;66
00C5 F523        dw      9205    ;67
00C7 3824        dw      9272    ;68
00C9 7824        dw      9336    ;69
00CB B524        dw      9397    ;70
00CD EF24        dw      9455    ;71
00CF 2725        dw      9511    ;72
00D1 5B25        dw      9563    ;73
00D3 8D25        dw      9613    ;74
00D5 BB25        dw      9659    ;75
00D7 E725        dw      9703    ;76
00D9 1026        dw      9744    ;77
00DB 3526        dw      9781    ;78
00DD 5826        dw      9816    ;79
00DF 7826        dw      9848    ;80
00E1 9526        dw      9877    ;81
00E3 AF26        dw      9903    ;82
00E5 C526        dw      9925    ;83
00E7 D926        dw      9945    ;84
00E9 EA26        dw      9962    ;85
00EB F826        dw      9976    ;86
00ED 0227        dw      9986    ;87
00EF 0A27        dw      9994    ;88
00F1 0E27        dw      9998    ;89
00F3 1027        dw      10000   ;90 degrees
                 ;
                 end
```

End Listing

by Michael Wiesenberg

A Potpourri of Good-Priced Hardware

The three-inch Winchesters are coming! One of the first is from SyQuest Technology. Their **SQ306** they claim is "the industry's first 3.9-inch (100mm) removable cartridge Winchester disk drive." SyQuest is aiming at the OEM market, but the price of $400 per unit in large quantities means we can expect to see these five-megabyte Winchesters arrive reasonably priced. The cartridges will sell in quantity for $30 each. The small size of this unit (1.625 inches tall) allows two SyQuest drives to fit into the same space as one 5.25-inch Winchester or mini-floppy drive. We should see these units integrated, probably stacked two at a time, into the next generation of portable computers. **Reader Service No. 306.**

Monitor the status of the "seven most important" (we are told) RS-232 lines with, what else?, **RS-232 Tester** from B & B Electronics. A female 25-pin connector at one end and a male connector at the other let you attach the Tester directly to the interface and leave it permanently in the line, with LEDs constantly displaying status, but not interfering with data transfer ability. $39.95 postpaid. **Reader Service No. 316.**

The **SSB-MPF Speech Synthesizer Board** from Etronix provides a 400-word vocabulary for the MPF-1 Micro-Professor, the Z-80-based single-board computer-in-a-book described in a previous column. The board does its input/output with the Micro-Professor's keyboard and speaker, and features a 4K time clock and speech utility EPROM, two EPROM sockets that can be used to expand the vocabulary, adjustable pitch and volume controls, a power adaptor, all necessary connection accessories and a manual. $129. **Reader Service No. 326.**

A Gallimaufry of Inexpensive Software

AUTODIFF is a file difference detector for CP/M by the Software Toolworks. Not only does this comparison utility list differences between files or produce a copy of the file with all changes flagged, it also reports insertions, deletions and changes, listing them in ASCII or hexadecimal format to a terminal, printer or disk file. Filters can be set to ignore non-printing characters or to display control characters. AUTODIFF can be used on most files and costs $29.95 (plus $2 postage and handling) on 5-inch disk for Osborne I and Heath/Zenith, and 8-inch CP/M disk. **Reader Service No. 346.**

Disk-Edit is a screen-oriented disk editor from Supersoft for CP/M programs. It can call up files that, according to Supersoft, "are not even accessible with a normal text editor, then edit those files in either ASCII or hexadecimal notation." The program loads a one-kilobyte section of a disk (hard or floppy) into a buffer, and then displays a dual-view "window" into the buffer. On the left are the hexadecimal values of each byte; on the right, the ASCII. Change either, and the corresponding value changes instantly in the other. Simple commands move the cursor up, down, left, right, to the next page or screen, to the start of the file and so on. There are also string searches and many other functions. Disk-Edit has a "terminal definition package" and can be configured to most CP/M systems. $100, or $15 for the manual only. **Reader Service No. 356.**

You probably thought that a text editor for the IBM Personal Computer would be expensive, but **WINDOW**, a full-screen text editor program from Intellect Associates Inc., costs $150. It uses all the screen and keyboard capabilities of the PC, including single-stroke editing commands with the function keys. It's not quite a word processor, but WINDOW does most of the things a word processor can, such as easily move the cursor, scroll in four directions, do global search and replace, insert and delete characters and lines, move, copy, split and join lines and edit text files larger than available memory. WINDOW runs under IBM-DOS, comes on 5.25-inch diskette, with documentation, and requires 64K, one drive, and a monochrome display and adapter. To turn WINDOW into a real word processor, add Intellect's **PCTEXT** for $100, and get a text processing package that indents, centers, controls line spacing and margins, justifies, inserts headings and footings, numbers pages, underlines and merges documents. It, too, runs under IBM-DOS, requires 48K, one drive and a printer. And while we're talking about Intellect Associates, they offer for the same configuration a data management system, **DMS**, a self-prompting, menu-driven system with which users create data entry forms on the screen in user-specified formats, and then enter, retrieve, modify, correct and delete data with the forms and print reports. The sequential ASCII data base files can be accessed by any language. DMS is written in **C88** (Intellect Associates' one-pass-compiler subset of C that costs $250, generates compact 8088 machine code and comes with a linker), which makes it run, they say, very fast, and also makes good use of the PC's screen and keyboard features. It requires 48K, one drive (two recommended) and monochrome display adapter. **Reader Service No. 366.**

Cross-assemblers can cost a bundle, but here's one for the Zilog Z8 that costs $150. **SYSTEM-Z8**, by Allen Ashley, for CP/M, includes a down-loader for the Zilog Z8 Development Module to transmit developed programs for in-circuit test. You get a macro-assembler with macro and conditional assembly and chaining, interactive editor/assembler, text editor, cross-reference generator, complete documentation and user support by mail or phone. Current SYSTEM-Z8 owners can phone for a free update. **Reader Service No. 376.**

Ensign Software offers a whole stack of reasonably-priced software for the IBM PC. **String Sort** is a machine-language sort routine for BASIC programs that sorts 1000 variable-length strings in less than four seconds, and 5000 three-character strings in fourteen. $24.95. **Electronic Disk** uses RAM as an "electronic disk drive" and printer spooler. The entire contents of a drive are dumped

to the memory-resident electronic disk drive, so that disk access programs do superfast I/O. You can also use the computer for other tasks while printing large files. $69.95. **ISAM Database** accesses up to seven data files simultaneously by full or partial keys using the *I*ndexed *S*equential *A*ccess *M*ethod, and is easily called from BASIC. It sorts 5000 records in twelve seconds. A demonstration of ISAM Database in the form of a complete name and address program is included in the $69.95 price. **Cross Reference** gives a complete cross-reference listing of a BASIC program saved in either ASCII or binary format. The listing includes all variables alphabetically with all line number references, line numbers referenced by other statements and all line numbers with PRINT and LPRINT. $24.95. All programs require one drive, DOS, disk BASIC, either monochrome or color/graphics and 48K (except Electronic Disk, 128K). **Reader Service No. 386.**

Central Point's real bargain is their **FILER** disk utility for drives of any number of tracks. It checks drive speed, tests the drive, copies a disk in thirty-five seconds; and has a file manager that catalogs or copies files, disks and DOS; deletes, locks and unlocks files; and changes the name and file type of the boot program, all for $19.95. **Reader Service No. 406.**

DDJ

JANUS

ADA*

Question: When can I use Ada?

Answer: NOW — with JANUS.

Ada is available now for your micro-computer.

JANUS is a subset of **Ada** which includes those features sorely missed by programmers on micros. Here is a list to help you decide for yourself.

+ Modular Separate Compilation
+ Single and Double Precision Floating Point Numbers
+ Binary Coded Decimal (BCD) Fixed Point Numbers
+ Integer and Long Integer Numbers
+ Complete String Handling
+ Sequential and Random Access I/O
+ Full Dynamic Allocation and Deallocation
+ Friendly Error Handling
+ An Assembler for interfacing assembly routines
+ A Linker for combining modules
+ True native code is produced
+ ROMable, reenterant code
+ Run-time library source code
+ Low Cost. **JANUS** is more cost effective than any other comparable Ada package
+ Inexpensive Updates
+ No royalties for programs written in **JANUS**
+ No hassle customer service

The language that is based on the past but looks to the uses of the future.

In short, all pluses. **JANUS** contains everything you need to do fast, structured program development in a micro environment.

JANUS is available for the CP/M, CP/M-86, and MS-DOS operating systems.

Now you too can take a step forward into the future on these computers:

8080/Z80 based systems: (All CP/M) Apple Softcard, North Star, Cromemco, Superbrain, TRS-80 Model II, and all CP/M 8″ disk systems.

8086 based systems: IBM Personal Computer, Victor 9000, Seattle Computer System II, Tecmar, Lomas Data Products, and all CP/M 8″ disk systems.

8080 or Z80, CP/M (requires 56K memory) — $300.00

8086/8088, CP/M-86 or MS-DOS (requires 96K memory) — $400.00

RR SOFTWARE

specialists in state of the art programming

P.O. BOX 1512 MADISON, WISCONSIN 53701 (608) 244-6436

Circle **no. 12** on reader service card.

Clinic *(Continued from page 9)*

well, but we'd rather see some evidence. Surely somebody out there has a PC, a suspicious mind, experience with the typical output of translator programs, and a few hours to spare. The MSDOS command DEBUG will disassemble storage. Use Ctrl-PrtSc to start a log of its output on the printer.

Actually, there are very good reasons why Microsoft would translate an existing interpeter rather than write a new one. Their MBASIC is a mature, well-debugged piece of code. Translation would let them produce a reliable product quickly and cheaply. Reliability, quick development, and low cost are three qualities that are conspicuously absent when you build a large piece of software from scratch. One can see how the "technical" consideration of execution speed could easily take a back seat to these desirable "management" criteria. . . .

Burke Smith of Shawnee Mission, KS, has found a bug in that mature product, MBASIC. It only seems to surface in version 5.1 of BASIC-80, and in IBM version D1.00. Try this sequence:

```
LET A#=0
LET A#=-A#
PRINT INT(A#),FIX(A#)
```

Smith claims that in the noted versions (but not in the more common versions 4.51 and 5.2) two answers of –1 will be diplayed, "rather than the zeroes that any reasonable person would expect."

Smith goes on to blame the problem on Microsoft's choice of binary floating-point rather than decimal float using a BCD representation. We think that's a red herring. We've seen a lot of misguided complaints along this line; the writers seem to think that there is something peculiar about the binary number system that makes it more prone to truncation errors ("round-off errors") than decimal is. Tain't so; a digit is a digit, no matter what your radix. The problem is the limited precision of the representation, not the number base. A decimal representation of a real number is just as prone to truncation errors as a binary, octal or hexadecimal one is.

There are standard techniques that will mitigate the effect of truncation errors. Hardware designers can incorporate an extra digit of precision in the temporary results held in machine registers, and round any bits that turn up in such a "guard digit" back into the result before storing the result. Few people remember now, but the initial design for the IBM 360 line lacked guard digits in the floating-point ALU; the first machines had them added in the field as engineering changes. Good scientific calculators get the same effect by carrying several low-order digits that don't show in the display. MBASIC could have achieved the same effect in software; we don't know if it tries to.

A decimal representation is better in just one way: it will get truncation errors on the numbers that people expect it to. Nobody is too surprised when the expression ((1/3)*3) does not yield a result of 1.0; most people can picture the string 0.333333 . . . being truncated and multiplied by 3 to yield 0.999999. The trouble with binary (or any other radix) is that some fractions that have finite representations in decimal become repeating "decimals" in another radix (and vice versa). When the system converts such a constant to its internal radix, it loses some precision before any calculations are done at all. For commercial arithmetic, the only answer is to use a fixed-point decimal representation like the ones available in PL/I and COBOL compilers. Floating-point arithmetic should *never* be used for commercial calculations, no matter what radix it is based on.

John Palmer of Boonville, CA, is grumpy about his new CP/M-86 system, which he procured from Godbout along with the Godbout 8085/8088 dual processor board. "The Godbout hardware is very good," he writes. "If you buy all the stuff from Godbout it will run." However, problems arose when he had to modify the Godbout BIOS. The first problem was that "the BIOS is very hard

SUBMIT.COM

```
; A patch to SUBMIT.COM which forces it to open an existing
; $$$.SUB file for appending, rather than erasing it.  The
; CCP reads that file in reverse order.  Thus a SUBMIT within
; a submitted file will "push" new records onto the end of the
; file, from whence the CCP will "pop" them.  The result is
; correct execution of nested SUBMIT commands.
;
subfcb  equ     05BBh    ; FCB for $$$.SUB
BDOS    equ     0005h
OPEN    equ     0211H    ; open-file subroutine of SUBMIT
;
        org     022Dh    ; the erase-file routine of SUBMIT
;
; It would seem that this location is called when an attempted
; open of $$$.SUB succeeds, showing that the file exists.
; Previously, this code would have erased the file.
;
opsl    lda     subfcb+15 ; open ok if extent not full
        ral               ; extent full if record count=>80h
        rnc               ; exit if count < 80h
        lxi     h,subfcb+12 ; extent number in FCB
        inr     m         ; try the next extent
; Code to invoke or re-invoke the create-next-extent routine
; of SUBMIT--which will call "opsl" above if the extent exists?
ops     lxi     d,subfcb ; open first (next?) extent
        jmp     create
;
; Routine to create the $$$.SUB file
;
        org     025Dh
create  call    OPEN     ; test $$$.SUB by opening it?
        inr     a        ; FFh -> 00h if open fails
        jnz     opsl     ; do above code if file exists
        lxi     d,subfcb ; file does not exist, make it
        mvi     c,22
        call    BDOS
        adi     01h      ; set carry flag if retcode=FFh
        ret
;
; Replacement code for original logic to open $$$.SUB
;
        org     04FEh
        call    ops      ; open last extent of $$$.SUB
        jc      0517h    ; error if open failed
        lda     subfcb+15 ; use the FCB "record count"
        sta     subfcb+32 ; ..to set the "current record"
        jmp     051Dh    ; continue with original code?
;
        end
```

to modify without the Sorcim anti-Intel ACT assembler." That would make us grumpy, too. There is no excuse for distributing a BIOS that requires an assembler that isn't part of the distributed system. "The source code for the BIOS is on the diskette," Palmer reports, "but you cannot use it until you buy Sorcim's ACT-86 for $175." Palmer thinks this is a tacky way of selling assemblers, and we agree.

Passing the Hat

We're scraping the bottom of the mailbag again, readers. You've all been very flattering in your praise of this column, but you aren't *contributing*. We need questions, like Knipp's. Pitfalls and Warnings, like Palmer's. Bugs, like Barker's. Patches, like Pasky's. Discoveries, like Hammond's. Puzzles. Confusions. Grumps. Cheap software. *Material!* Please?

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 526**

Letters *(Continued from page 6)*

by *DDJ* readers. First, however, I attacked its primary problem: it's *slow*.

I studied the code for a while, and decided to change the global symbol table from a linearly searched list to a hash table. I also put the macros in the global symbol table, with pointers into the array of macro definitions. Since the original code was well-structured, changing algorithms was not difficult.

I changed "numglbs" from 300 to 512 (a power of two), "SYMTBSZ" from 5040 to 8008, and introduced "#define MASKGLBS 511". I introduced "#define MACRO 5", giving a new possible entry for "ident". Since I'm putting the macro names into the global symbol table rather than the macro table, I changed "macqsize" from 1000 to 500. "macptr" has to be initialized to 1 instead of 0. (Initializing it to 0 causes a subtle bug affecting only the first macro. I'll leave the details as an exercise for the student.)

I added the following initializing code to main():

```
        glbptr=STARTGLB;
        while(glbptr<ENDGLB){
                *glbptr=0;
                glbptr=glbptr+SYMSIZ;
        }
        glbptr=STARTGLB+SYMSIZ*5;
                /* clear global symbols
 */
```

"dumpglbs()" needed the declaration "int i;", and the top of the loop changed to

```
    i=NUMGLBS;
    while(i--){  /* 6/19/82  jrvz */
        if(*cptr){
            if((cptr[ident]!=function)
            &(cptr[ident]!=MACRO))
            /* do if anything but
            function or macro
            jrvz 6/19/82 */
```

I introduced a simple hash function:

```
hash(sname)
    char *sname;
{   int h,c;
    h=*sname;
    while(c=*(++sname)) h=(h<<1)+c;
    return h;
}
```

I then rewrote the following routines where boxes indicate changes):

```
findglb(sname)
     /* cptr is set to entry if found,
     or appropriate empty slot if not */
     char *sname;
{    int h;
     h=hash(sname)&MASKGLBS;
     cptr=STARTGLB+h*SYMSIZ;
     while(0==
     astreq(sname,cptr,namemax)){
          if(*cptr==0) return 0;
          cptr=cptr+SYMSIZ;
          if(cptr==ENDGLB)cptr=STARTGLB;
     }
     return cptr;
}
```

```
addglb(sname,id,typ,value)
     char *sname,id,typ;
     int value;
{    char *ptr;
     if(findglb(sname))return cptr;
     if(glbptr>=ENDGLB)
          { error("global symbol table
overflow");
          return 0;
          }
     ptr=cptr;
     while(an(*ptr++ = *sname++));
                    /* copy name */
     cptr[ident]=id;
     cptr[type]=typ;
     cptr[storage]=statik;
     cptr[offset]=value;
     cptr[offset+1]=value>>8;
     glbptr=glbptr+SYMSIZ;
     return cptr;
}
```

```
addmac()
{    char sname[namesize];

     if(symname(sname)==0)
          {illname();
          kill();
          return;
          }
     addglb(sname,MACRO,0,macptr);
     while(ch()==' ' | ch()==9)
          gch();
     while(putmac(gch()));
     if(macptr>=macmax)
          error("macro table full");
     }
```

```
findmac(sname)
          /* rewritten 6/19/82   jrvz */
     char *sname;
{    if((findglb(sname)!=0)&
     (cptr[ident]==MACRO))
          {return((cptr[offset]&255)+
          (cptr[offset+1]<<8));
          }
     return 0;
}
```

As a result of these changes, the compiler speed has more than doubled. I also enlarged the disk buffers from 128 bytes to 512 bytes. That sped compilation up a further 20%, to about 220 lines/minute on my 2.5 MHz Z-80.

Has anyone installed floating-point variables? Does anyone have a C interpreter, that would let me debug my code quickly? (Note that Tiny-c accepts a different language.)

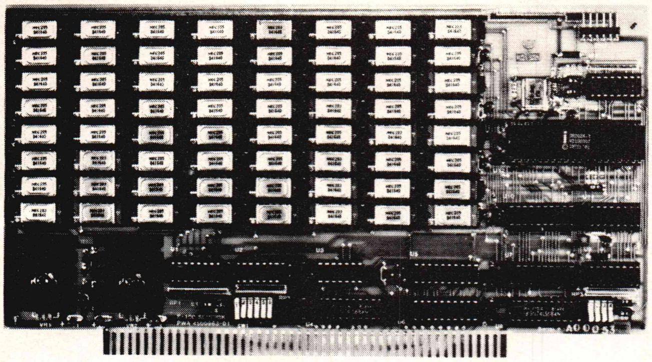
Keep the Small-C articles coming!

Yours,
James R. Van Zandt
26 Shelton St.
Nashua, NH 03062

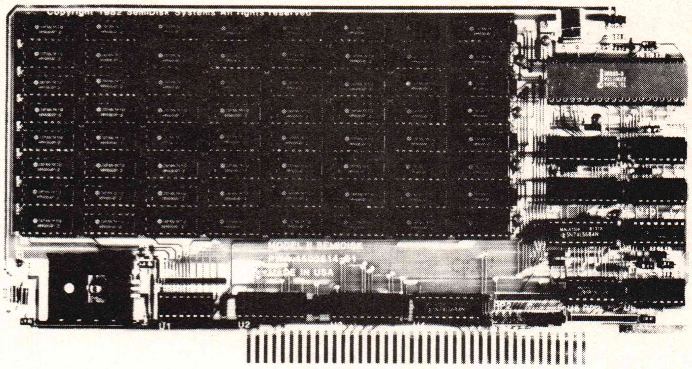
Editor's Note: Readers who are following the developments in Small-C, which originated with Ron Cain's Small-C compiler, should keep a close eye on DDJ *in the coming months. We have a number of exciting articles and listings scheduled.*

DDJ

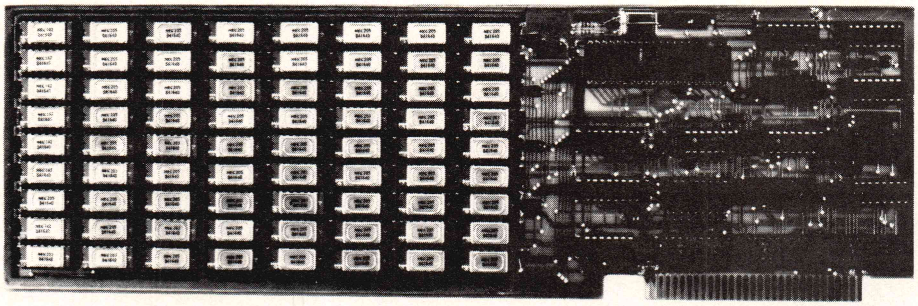
A FULL LINE OF SEMIDISKS



S-100



TRS 80 Model 2



IBM Personal Computer

Do you use your computer? Or does your computer "use" you? Face it, if you're using floppies, your time is being wasted. Because a floppy is an inefficient random access storage device. Each time the processor wants to transfer data, it has to wait an eternity for the disk to rotate and the head to move.

So what do you do? Get a SemiDisk, quick. It's a large capacity semiconductor memory board that is driven by software to operate like a disk drive. Without all the waiting. Do everything you'd do on a floppy or hard disk, with no modifications to your software or hardware. Two board sizes are available: 512K and 1 Megabyte. (the highest density microcomputer memory board in the world) And you can put up to 8 megabytes in a system by adding more storage boards.

What do you need to use it? Just an S-100 system with CP/M 2.2. Or a TRS-80 Model 2 system with CP/M 2.2. Or an IBM Personal Computer. That's it. No special processors, DMA, I/O, or disk controllers are required. Plug it in and run the installation program, and you're on your way. Fast! Even better, we supply full source code to the driver software, in case you'd like to do your own interfacing.

Best of all, the SemiDisk's price won't warp your wallet. Compare specs, cost/megabyte, storage capacity, and compatibility with the competition. You'll see that the SemiDisk is a disk emulator truly worthy of the name. SemiDisk has battery-backup capability, too.

Consider our limited warranty: A full year, covering all parts and labor. Consider our liberal 15 day return policy. Price? $1995 for 512K byte SemiDisk, $2995 for 1 Megabyte SemiDisk. Both from stock. $10.00 for manual. VISA, Mastercard, COD orders accepted. Dealer and OEM inquiries welcomed. (Specify system type and disk format when ordering.)

Someday, you'll get a SemiDisk.
Until then, you'll just have to . . . . wait.

NO WAITING

T.M.

SemiDisk SYSTEMS
P.O. Box GG
Beaverton, OR 97075
(503)-642-3100

Call (503)-646-5510 for CBBS®/NW, a Semi-Disk-equipped computer bulletin board.
SemiDisk trademark of SemiDisk Systems; TRS-80 trademark of Radio Shack

Circle no. **168** on reader service card.

ADVERTISERS INDEX


CB80
Ultra FAST BASIC Compiler
CB80,™ Compiler System's new native code Basic compiler, offers maximum speed and flexibility in creating applications to solve today's business problems.
CBASIC™ compatible: As an addition to the CBASIC family, CB80 has all the features of CBASIC (14 digit accuracy, long variable names, stream and record I/O, multiple line functions) plus these extras:
• Relocatable machine code • 32K byte strings • Nested IF statements • ON ERROR GOTO • Variable type declarations • CALL statement with parameters • EXTERNAL and PUBLIC functions • Local variables in functions • Alphanumeric labels • Record LOCK and UNLOCK
Expand your versatility. CB80 includes our LK80™ linker. It allows you to create programs in separate modules and easily combine them. Powerful CHAINING capabilities, multiple library scanning, and easy linkage to assembly routines, are all part of LK80.
CP/M® and MP/M II™ compatible: CB80 supports the popular CP/M and MP/M II operating systems. CB80's record LOCK and UNLOCK, combined with its superior speed makes it a natural for multiuser environments.
Increase productivity and profits. Faster execution boosts system throughput and maximizes your computer's resources—an essential user feature. Coupled with reduced programming time, CB80 definitely improves your bottom line benefits.
For your free CB80 brochure and licensing details, call us at (408) 649-3896, or write us today.
DIGITAL RESEARCH®
P.O. Box 579, Pacific Grove, California 93950
Europe: Vector, Int'l., Leuven, Belgium, 32(16)202496
Far East: Microsoftware Assoc., Tokyo, Japan, 03-403-2120
CBASIC™
Solving complex business problems with BASIC simplicity
CB80, CBASIC, LK80 and MP/M II are trademarks of Digital Research. CP/M is a registered trademark of Digital Research. © Copyright 1981 Digital Research
Circle no. 233 on reader service card.